Atria Institute of Technology

Department of Information Science and Engineering

Bengaluru-560024

ACADEMIC YEAR: 2021-2022

EVEN SEMESTER NOTES

Semester : 6th Semester

Subject Name : Web Technology and its Applications

Subject Code : 18CS63

Faculty Name : Mrs. Shruthi B

MODULE1

Chapter1: Introduction to HTML

1 What is HTML and Where Did It Come From?
2 HTML Syntax
3 Semantic Markup
4 Structure of HTML Documents
5 Quick Tour of HTML Elements
6 HTML5 Semantic Structure Elements.

What Is HTML and Where Did It Come from?

- HTML is defined as a **markup language**. A markup language is simply a way of annotating a document in such a way as to make the annotations distinct from the text being annotated.
- Markup languages such as HTML, Tex, XML, and XHTML allow users to control how text and visual elements will be laid out and displayed. The term comes from the days of print, when editors would write instructions on manuscript pages that might be revision instructions to the author or copy editor.
- At its simplest, **markup** is a way to indicate *information about the content* that is distinct from the content. This "information about content" in HTML is implemented via **tags.**
- Markup languages are able to encode information how to display the content for the end user.
- The combining semantic markup with presentation markup is no longer permitted in HTML5, "formatting the content" for display remains a key reason why HTML was widely adopted.

XHTML

- Instead of growing HTML, the W3C turned its attention in the late 1990s to a new specification called XHTML 1.0, which was a version of HTML that used stricter **XML** (extensible markup language) syntax rules.
- The goal of XHTML with its strict rules was to make page rendering more predictable by forcing web authors to create web pages without **syntax errors**.

- To help web authors, two versions of XHTML were created: **XHTML 1.0Strict** and **XHTML 1.0 Transitional**.
    - The **strict** version was meant to be rendered by a browser using the strict syntax rules and tag support described by the W3C XHTML 1.0 Strict specification.
    - The **transitional** recommendation is a more forgiving flavour of XHTML, and was meant to act as a temporary transition to the eventual global adoption of XHTML Strict.

XML

- XML is a more general markup language than HTML. It is used to mark up any type of data. XML-based data formats (called **schemas** in XML) are almost everywhere.
- The following is an example of a simple XML document:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<art>
  <painting id="290">
    <title>Balcony</title>
    <artist>
      <name>Manet</name>
      <nationality>France</nationality>
    </artist>
    <year>1868</year>
    <medium>Oil on canvas</medium>
  </painting>
</art>
```

- The XML-based syntax rules for XHTML are pretty easy to follow. The main rules are:
    - There must be a single root element.
    - Element names are composed of any of the valid characters (most punctuation symbols and spaces are not allowed) in XML.
    - Element names can't start with a number.
    - Element and attribute names are case sensitive.
    - Attributes must always be within quotes.
    - All elements must have a closing element (or be self-closing).

- XML also provides a mechanism for validating its content. It can check, for instance, whether an element name is valid, or elements are in the correct order, or that the elements follow a proper nesting hierarchy.
- **HTML validators** means verifying that a web page's markup followed the rules for XHTML Transitional or Strict. Web developers often placed proud images on their sites

to tell the world at large that their site followed XHTML rules (and also to communicate their support for web standards).

- Backwards compatibility with HTML and XHTML 1.0 was dropped. Browsers would become significantly less forgiving of invalid markup.

Figure 1.1 W3C XHTML validation service

HTML5

- A group of developers at Opera and Mozilla formed the **WHATWG** (Web Hypertext Application Technology Working Group) group within the W3C.
- There are three main aims to HTML5:
    1. Specify unambiguously how browsers should deal with invalid markup.
    2. Provide an open, nonproprietary programming framework (via JavaScript) for creating rich web applications.
    3. Be backwards compatible with the existing web.
- HTML in HTML5 is now a living language: that is, it is a language that evolves and develops over time. As such, every browser will support a gradually increasing subset of HTML5 capabilities.

HTML Syntax

Elements and Attributes

- HTML documents are composed of textual content and HTML elements.
- The term **HTML element** is often used interchangeably with the term **tag**.
- An HTML element is identified in the HTML document by tags.
- A tag consists of the element name within angle brackets.
- The element name appears in both the beginning tag and the closing tag, which contains a forward slash followed by the element's name, again all enclosed within angle brackets.
- The closing tag acts like an off-switch for the on-switch that is the start tag.
- HTML elements can also contain attributes. An **HTML attribute** is a name=value pair that provides more information about the HTML element.
- In XHTML, attribute values had to be enclosed in quotes; in HTML5, the quotes are optional.
- Some HTML attributes expect a number for the value. These will just be the numeric value; they will never include the unit.

Figure 1.2 The parts of an HTML element

- Figure 1.2 illustrates the different parts of an HTML element, including an example of an empty HTML element.
- An **empty element** does not contain any text content, instead, it is an instruction to the browser to do something. Perhaps the most common empty element is <img>, the image element.
- In XHTML, empty elements had to be terminated by a trailing slash, the trailing slash in empty elements is optional.

Nesting HTML Elements

- HTML element will contain other HTML elements. The container element is said to be a parent of the contained, or child, element.
- Any elements contained within the child are said to be **descendants** of the parent element; likewise, any given child element, may have a variety of **ancestors**.
- In XHTML, all HTML element names and attribute names had to be lowercase.
- HTML5 and HTML 4.01 does not care whether you use upper or lowercase for element or attribute name.

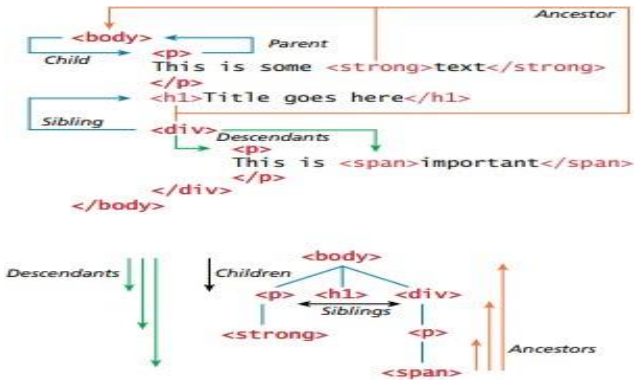


Figure 1.3: HTML document outline

- This underlying family tree or hierarchy of elements **Cascading Style Sheets** (CSS) and JavaScript programming and parsing.
- In order to properly construct this hierarchy of elements, our browser expects each HTML nested element to be properly nested. That is, a child's ending tag must occur before its parent's ending tag, as shown in Figure 1.4

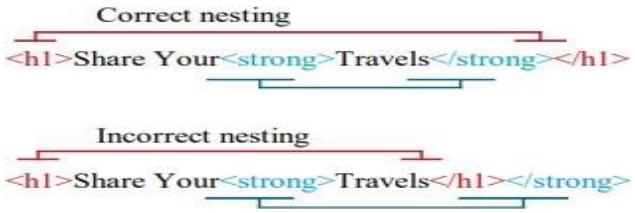


Figure 1.4: The proper nesting of HTML elements

Semantic Markup

- A strong and broad consensus has grown around the belief that HTML documents should **only** focus on the structure of the document.

- Information about how the content should look when it is displayed in the browser is best left to CSS (Cascading Style Sheets).
- Beginning HTML authors are often counseled to create **semantic HTML** documents. That is, an HTML document should not describe how to visually present content, but only describe its content's structural semantics or meaning.
- That structure (and to be honest the presentation as well) makes it easier for the reader to quickly grasp the hierarchy of importance as well as the broad meaning of the information in the document.
- Structure is a vital way of communicating information in paper and electronic documents that are used to describe the basic structural information in a document, such as headings, lists, paragraphs, links, images, navigation, footers, and so on.

Advantages

1. **Maintainability**. Semantic markup is easier to update and change than webpages that contain a great deal of presentation markup.
2. **Faster**. Semantic web pages are typically quicker to author and faster to download.
3. **Accessibility**. Not all web users are able to view the content on web pages. Users with sight disabilities experience the web using voice reading software.
    - Visiting a web page using voice reading software can be a very frustrating experience if the site does not use semantic markup.
    - For instance, the United States government has its own Section 508 Accessibility Guidelines (**http://www.section508.gov**).
4. **Search engine optimization**. For many site owners, the most important users of a website are the various search engine crawlers. Semantic markup provides better instructions for these crawlers: it tells them what things are important content on the site.

Structure of HTML Documents

- The <title> element (Item **1** in Figure 1.5) is used to provide a broad description of the content. The title is not displayed within the browser window. Instead, the title is typically displayed by the browser in its window and/or tab, as shown in the example in Figure 1.5.

- The title is used by the browser for its bookmarks and its browser history list. The operating system might also use the page's title, for instance, in the Windows taskbar or in the Mac dock. And even search engines will typically use the page's title as the linked text in their search engine result pages.
- Figure 1.6 illustrates a more complete HTML5 document that includes these other structural elements as well as some other common HTML elements.

```
<!DOCTYPE html>
<title>A Very Small Document</title>
<p>This is a simple document with not much content</p>
```

Figure 1.5: One of the simplest possible HTML5 documents

DOCTYPE

- Item **1** in Figure 1.6 points to the DOCTYPE (short for **Document Type Definition**) element, which tells the browser what type of document it is about to process.
- In HTML5 doctype is quite short in comparison to one of the standard doctype specifications for XHTML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

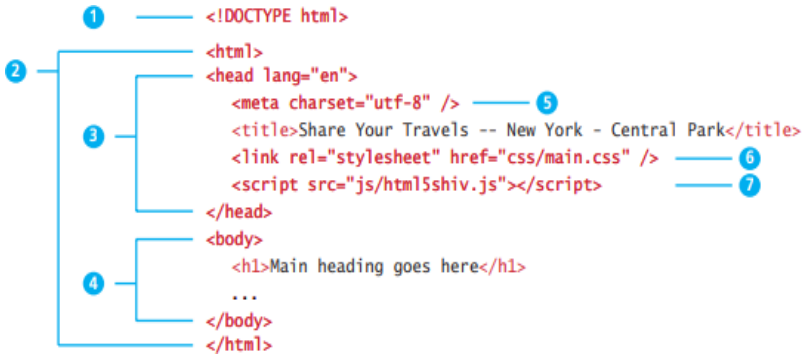


Figure 1.6: Structure elements of an HTML5 document

- The doctype was used to tell the browser to render an HTML document.
- Document Type Definitions (DTD) define a document's type for markup languages such as HTML and XML.
- In both these markup languages, the DTD must appear near the beginning of the document. DTDs have their own syntax that defines allowable element names and their order.

Head and Body

- HTML5 does not require the use of the <html>, <head>, and <body> elements. However, in XHTML they were required, and most web authors continue to use them.
- The <html> element is sometimes called the **root element** as it contains all the other HTML elements in the document.
- It also has a lang attribute. This optional attribute tells the browser the natural language that is being used for textual content in the HTML document, which is English in this example.
- This doesn't change how the document is rendered in the browser; rather, search engines and screen reader software can use this information. HTML pages are divided into two sections: the **head** and the **body**, which correspond to the <head> and <body> elements.
- The head contains descriptive elements *about* the document, such as its title, any style sheets or JavaScript files it uses, and other types of meta information used by search engines and other programs.
- The body contains content (both HTML elements and regular text) that will be displayed by the browser.
- Character encoding refers to which character set standard is being used to encode the characters in the document.
- **UTF-8** is a more complete variable-width encoding system that can encode all 110,000 characters in the Unicode character set (which in itself supports over 100 different language scripts).
- Item 6 in Figure 1.6 specifies an external CSS style sheet file that is used with this document. Virtually all commercial web pages make use of style sheets to define the visual look of the HTML elements in the document.

- Styles can also be defined within an HTML document for consistency's sake, most sites place most or all of their style definitions within one or more external style sheet files.
- Finally, Item 7 in Figure 1.6 references an external JavaScript file. Most modern commercial sites use at least some JavaScript. Like with style definitions, JavaScript code can be written directly within the HTML or contained within an external file.

Quick Tour of HTML Elements

Headings

- Item 1 in Figure 1.7 defines two different headings. HTML provides six levels of heading (h1 through h6), with the higher heading number indicating a heading of less importance.
- In the real-world documents we know that headings are an essential way for document authors to show their readers the structure of the document.
- Headings are also used by the browser to create a **document outline** for the page. Every web page has a document outline. This outline is not something that we see.

```
<body>
  <h1>Share Your Travels</h1>
  <h2>New York - Central Park</h2>
  <p>Photo by Randy Connolly</p>
  <p>This photo of Conservatory Pond in
     <a href="http://www.centralpark.com/">Central Park</a>
     New York City was taken on October 22, 2015 with a
     <strong>Canon EOS 30D</strong> camera.
  </p>
  <img src="images/central-park.jpg" alt="Central Park" />

  <h3>Reviews</h3>
  <div>
     <p>By Ricardo on <time>September 15, 2015</time></p>
     <p>Easy on the HDR buddy.</p>
  </div>

  <div>
     <p>By Susan on <time>October 1, 2015</time></p>
     <p>I love Central Park.</p>
  </div>

  <p><small>Copyright &copy; 2015 Share Your Travels</small></p>
</body>
```

Figure 1.7: Sample HTML5 document

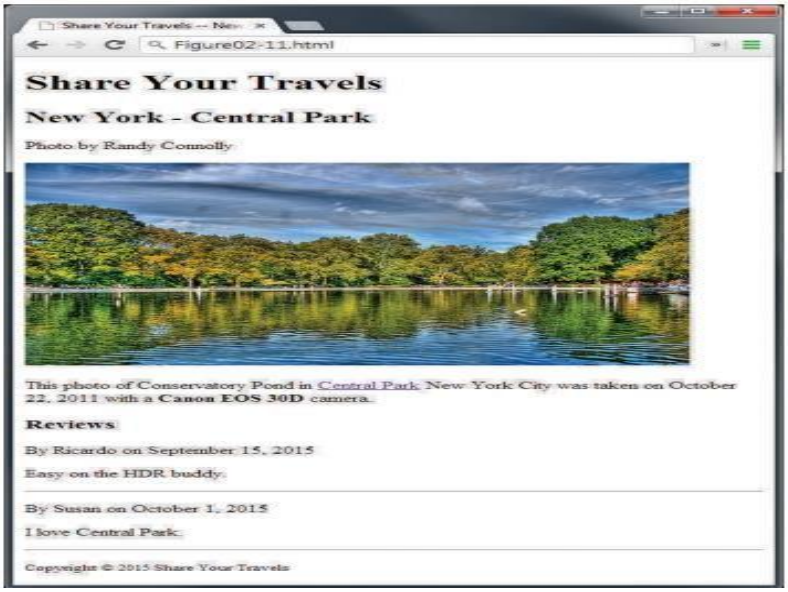


Figure 1.8: figure 1.7 in the browser

- This document outline is constructed from headings and other structural tags in our content. There is a variety of web-based tools that can be used to see the document outline.
- Specify a heading level that is semantically accurate; do not choose a heading level because of its default presentation. Rather, choose the heading level because it is appropriate (e.g., choosing <h3> because it is a third-level heading and not a primary or secondary heading).

Paragraphs and Divisions

- Item 2 in Figure 1.7 defines two paragraphs, the most basic unit of text in an HTML document.
- <p> tag is a container and can contain HTML and other **inline HTML elements** (the and <a> elements in Figure 1.7).
- The line break element (br /) forces a line break. It is suitable for text whose content belongs in a single paragraph but which must have specific line breaks: for example, addresses and poems.
- Item 6 in Figure 1.7 illustrates the definition of a element. This element is also a container element and is used to create a logical grouping of content (text and other HTML elements, including containers such as <p> and other elements). The

element has no intrinsic presentation; it is frequently used in contemporary CSS-based layouts to mark out sections.

Links

- Item 3 in Figure 1.7 defines a hyperlink. Links are an essential feature of all web pages. Links are created using the <a> element (the "a" stands for anchor).
- A link has two main parts: the destination and the label. As can be seen in Figure 1.9, the label of a link can be text or another HTML element such as an image.

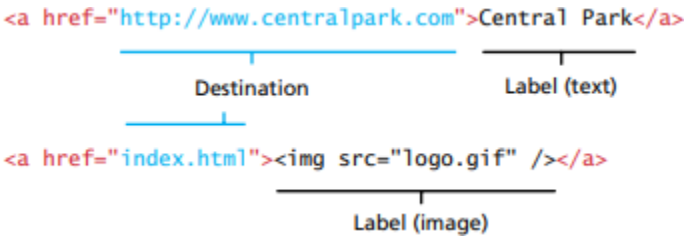


Figure 1.9: Two parts of a link

- We can use the anchor element to create a wide range of links. These include:
  - Links to external sites (or to individual resources such as images or movies on an external site).
  - Links to other pages or resources within the current site.
  - Links to other places within the current page.
  - Links to particular locations on another page (whether on the same site or on an external site).
  - Links that are instructions to the browser to start the user's email program.
  - Links that are instructions to the browser to execute a JavaScript function.
  - Links that are instructions to the mobile browser to make a phone call.
  - Figure 1.10 illustrates the different ways to construct link destinations

Figure 1.10: Different link destinations

URL Relative Referencing

- When referencing a page or a resource is on different site or different server then, it is referred as **absolute** referencing.
  - the protocol (typically, http://),
  - the domain name,
  - any paths, and then finally
  - the file name of the desired resource.
- When referencing a page or a resource is on the same site or same server as our HTML document, it is referred as **relative** referencing.
  - If the URL does not include the "**http://**" then the browser will request the current server for the file.
  - If all the resources for the site reside within the same **directory** (folder), then we can reference those other resources simply via their file name.

- However, most real-world sites contain too many files to put them all within a single directory. For these situations, a relative pathname is required along with the file name.
- The **pathname** tells the browser where to locate the file on the server. Pathnames on the web follow Unix conventions.
  - Forward slashes ("/") are used to separate directory names from each other and from file names.
  - Double- periods ("**..**") are used to reference a directory "above" the current one in the directory tree.

➢ Figure 1.11 illustrates the file structure of an example site. Table 1.1 provides additional explanations and examples of the different types of URL referencing.

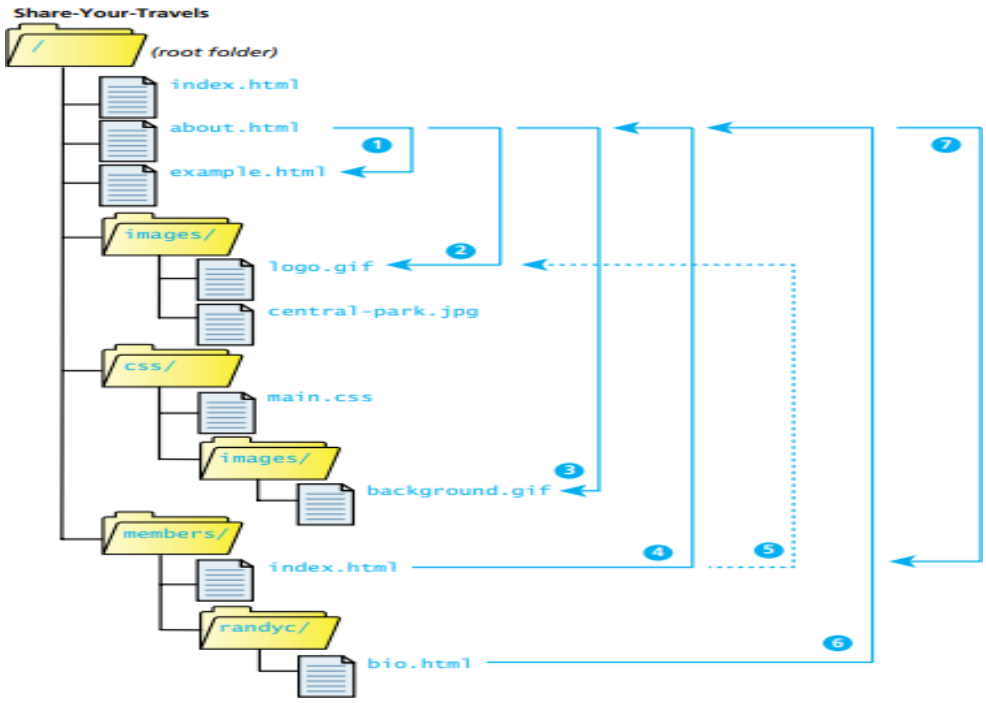


Figure 1.11: Example site directory tree

Inline Text Elements

➢ The html elements which do not disrupt the flow of text (i.e., cause a line break) are referred as inline elements. HTML defines over 30 of these elements. Table 1.2 lists some of the most commonly used of these elements.

Relative Link Type	Example
1 Same Directory To link to a file within the same folder, simply use the file name.	To link to example.html from about.html (in Figure 2.17), use: <a href="example.html">
2 Child Directory To link to a file within a subdirectory, use the name of the subdirectory and a slash before the file name.	To link to logo.gif from about.html, use: <a href="images/logo.gif">
3 Grandchild/Descendant Directory To link to a file that is multiple subdirectories *below* the current one, construct the full path by including each subdirectory name (separated by slashes) before the file name.	To link to background.gif from about.html, use: <a href="css/images/background.gif">
4 Parent/Ancestor Directory Use "../" to reference a folder *above* the current one. If trying to reference a file several levels above the current one, simply string together multiple "../".	To link to about.html from index.html in members, use: <a href="../about.html"> To link to about.html from bio.html, use: <a href="../../about.html">
5 Sibling Directory Use "../" to move up to the appropriate level, and then use the same technique as for child or grandchild directories.	To link to about.html from index.html in members, use: <a href="../images/about.html"> To link to background.gif from bio.html, use: <a href="../../css/images/background.gif">
6 Root Reference An alternative approach for ancestor and sibling references is to use the so-called root reference approach. In this approach, begin the reference with the root reference (the "/") and then use the same technique as for child or grandchild directories. **Note that these will only work on the server! That is, they will not work when you test it out on your local machine.**	To link to about.html from bio.html, use: <a href="/about.html"> To link to background.gif from bio.html, use: <a href="/images/background.gif">
7 Default Document Web servers allow references to directory names without file names. In such a case, the web server will serve the default document, which is usually a file called index.html (Apache) or default.html (IIS). **Again, this will only generally work on the web server.**	To link to index.html in members from about.html, use either: <a href="members"> Or <a href="/members">

Table 1.1: Sample Relative Referencing

Element	Description
<a>	Anchor used for hyperlinks.
<abbr>	An abbreviation
 	Line break
<cite>	Citation (i.e., a reference to another work).
<code>	Used for displaying code, such as markup or programming code.
	Emphasis
<mark>	For displaying highlighted text
<small>	For displaying the fine-print, i.e., "non-vital" text, such as copyright or legal notices.
	The inline equivalent of the element. It is generally used to mark text that will receive special formatting using CSS.
	For content that is strongly important.
<time>	For displaying time and date data

Table 1.2: Common Text-Level Semantic Elements

Images

- Item 5 in Figure 1.7 defines an image. While the <img> tag is the oldest method for displaying an image, it is not the only way. In fact, it is very common for images to be added to HTML elements via the background-image property in CSS, a technique.

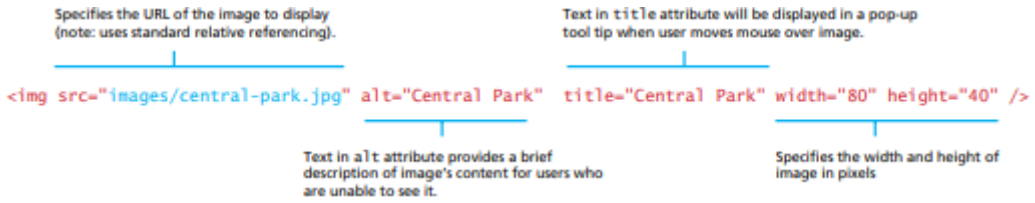


Figure 1.12: The <img> element

Character Entities

- Item 9 in Figure 1.7 illustrates the use of a **character entity**.
- These are special characters for symbols for which there is either no easy way to type them via a key board (such as the copyright symbol or accented characters) or which have are reserved meaning in HTML (for instance the "<" or ">" symbols).
- There are many HTML character entities which can be used in an HTML document by using the entity name or the entity number. Some of the most common are listed in Table 1.3.

Entity Name	Entity Number	Description
		Nonbreakable space. **The browser ignores multiple spaces in the source HTML file. If you need to display multiple spaces, you can do so using the nonbreakable space entity.**
&lt;	&#60;	Less than symbol ("<").
&gt;	&#62;	Greater than symbol (">").
&copy;	&#169;	The © copyright symbol.
&euro;	&#8364;	The € euro symbol.
&trade;	&#8482;	The ™ trademark symbol.
&uuml;	&#252;	The ü—i.e., small u with umlaut mark.

Table 1.3: Common Character Entities

Lists

A Collection of items forms a lists. HTML provides three types of lists:

1. **Unordered lists(ul)**. Collections of items in no particular order; these are by default rendered by the browser as a bulleted list.
2. **Ordered lists(ol)**. Collections of items that have a set order; these are by default rendered by the browser as a numbered list.
3. **Definition lists(dl)**. Collection of name and definition pairs. These tend to be used infrequently. The various list elements are container element containing list item elements (<li>). Other HTML elements can be included with in the <li> container, as shown in the first list item of the unordered list in Figure 1.13.

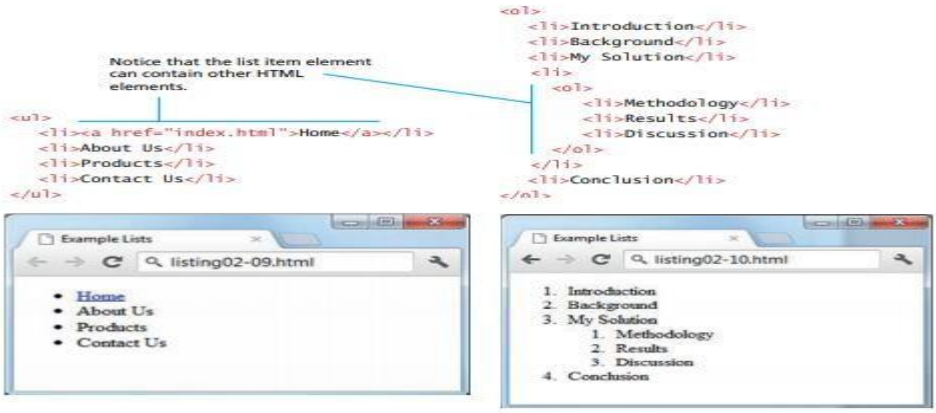


Figure 1.13: List elements and their default rendering

HTML5 Semantic Structure Elements

- Semantic structural elements describes its meaning to both the browser and the developer.
- Most complex websites are absolutely packed solid with elements. Most of these are marked with different id or class attributes.
- HTML5 proposes new semantic block structuring elements. The idea behind using these elements is that our markup will be easier to understand because we will be able to replace some of your sprawl with cleaner and more self-explanatory HTML5 elements.

Header and Footer

- Most website pages have a recognizable header and footer section. Typically the **header** contains the site logo and title, horizontal navigation links, and perhaps one or two horizontal banners.
- The typical **footer** contains less important material, such as smaller text versions of the navigation, copyright notices, information about the site's privacy policy, and perhaps twitter feeds or links to other social sites.

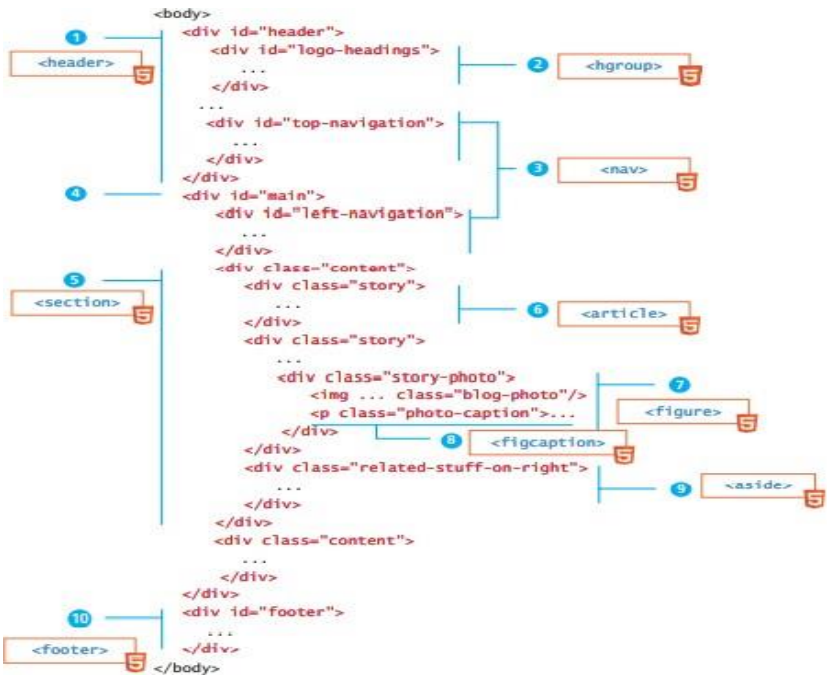


Figure 1.14: Sample based XHTML layout (with HTML5 equivalents)

- Both the HTML5 <header> and <footer> element can be used not only for *page* headers and footers, but also for header and footer elements within other HTML5 containers, such as <article> or <section>.
- The header element typically contains the headings for a section (an h1–h6 element or hgroup element), along with content such as introductory material or navigational aids for the section.
- The browser really doesn't care how one uses these HTML5 semantic structure elements. Just like with the element, there is no predefined presentation for these tags.

```
<header>
<img src="logo.gif" alt="logo" />
<h1>Fundamentals of Web Development</h1>
...
</header>
<article>
   <header>
      <h2>HTML5 Semantic Structure Elements</h2>
      <p>By <em>Randy Connolly</em></p>
      <p><time>September 30, 2015</time></p>
   </header>
   ...
</article>
```

Listing 1.1: Heading example

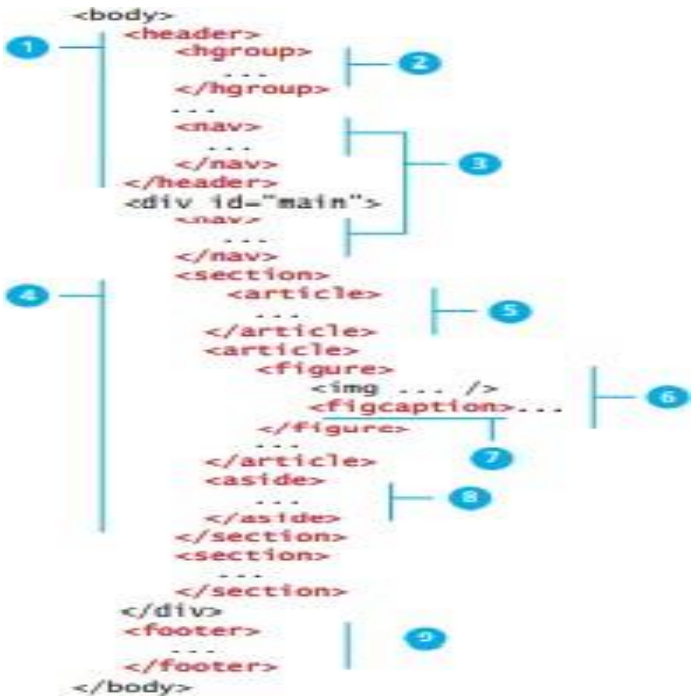


Figure 1.15: Sample layout using new semantic structure elements

Heading Groups

- The <hgroup> element (item 2 in Figure 1.15) can be used to group heading tags together within one container.
- The <hgroup> element can be used in contexts other than a header. For instance, one could also use an <hgroup> within an <article> or a <section> element as well.
- The <hgroup> element can *only* contain <h1>, <h2>, etc., elements. Listing 2.2illustrates two example usages of the <hgroup> element.

```
<header>
    <hgroup>
    <h1>Chapter Two: HTML 1</h1>
    <h2>An Introduction</h2>
    </hgroup>
</header>
<article>
    <hgroup>
    <h2>HTML5 Semantic Structure Elements</h2>
    <h3>Overview</h3>
    </hgroup>
</article>
```

Listing1.2: hgroup example

Navigation

- The <nav> element (item 3 in Figure 1.15) represents a section of a page that contains links to other pages or to other parts within the same page. The <nav> element was intended to be used for major navigation blocks.
- Not all groups of links on a page need to be in a nav element—the element is primarily intended for sections that consist of major navigation blocks.
- In particular, it is common for footers to have a short list of links to various pages of a site, such as the terms of service, home page, and a copyright page. The <footer> alone is sufficient for such cases; while a nav element such cases, it is usually unnecessary

```
<header>
  <img src="logo.gif" alt="logo" />
  <h1>Fundamentals of Web Development</h1>
  <nav role="navigation">
    <ul>
        <li><a href="index.html">Home</a></li>
        <li><a href="about.html">About Us</a></li>
        <li><a href="browse.html">Browse</a></li>
    </ul>
  </nav>
</header>
```

Listing 1.3: nav example

Articles and Sections

Articles

- The item 5 in figure 1.15 indicates the article element. The article element represents a section of content that forms an independent part of a document or site; for example, a magazine or newspaper article, or a blog entry.
- The article element represents a self-contained composition in a document, page, application, or site and that is, in principle, independently distributable or reusable, e.g. in syndication. This could be a forum post, a magazine or newspaper article, a blog entry, a user-submitted comment, an interactive widget or gadget, or any other independent item of content

Sections

- The item 4 in figure 1.15 indicates the section element. The section element represents a section of a document, typically with a title or heading.
- The section element represents a generic section of a document or application. A section, is a thematic grouping of content, typically with a heading. Examples of sections would

be chapters, the various tabbed pages in a tabbed dialog box, or the numbered sections of a thesis. A Website's home page could be split into sections for an introduction, news items, and contact information.

Figure and Figure Captions

- In HTML5 we can instead use the more obvious <figure> and <figcaption>elements (items 6 and 7 in Figure1.15).
- The element can be used not just for images but for any type of essential content that could be moved to a different location in the page or document and the rest of the document would still make sense.
- The figure element represents some flow content, optionally with a caption, that is self-contained and is typically referenced as a single unit from the main flow of the document.
- The element can thus be used to annotate illustrations, diagrams, photos, code listings, etc, that are referred to from the main content of the document, but that could, without affecting the flow of the document, be moved away from that primary content, e.g. to the side of the page, to dedicated pages, or to an appendix.

Figure 1.16: The figure and figcaption elements in the browser

Aside

- The <aside> element (item 8 in Figure 1.15) is similar to the <figure> element in that it is used for marking up content that is separate from the main content on the page.
- The <aside> element "represents a section of a page that consists of content that is tangentially related to the content around the aside element". The <aside> element could thus be used for sidebars, pull quotes, groups of advertising images, or any other grouping of non-essential elements.

Chapter 2: Introduction to CSS

1 What is CSS?
2 CSS Syntax
3 Location of Styles
4 Selectors
5 The Cascade: How Styles Interact
6 The Box Model
7 CSS Text Styling

What Is CSS?

- HTML should not describe the formatting or presentation of documents. Instead that presentation task is best performed using **Cascading Style Sheets (CSS)**.
- CSS is a W3C standard
  - For describing the appearance of HTML elements.
  - To describe CSS's function is to say that CSS is used to define the **presentation** of HTML documents.
- With CSS, we can assign font properties, colors, sizes, borders, background images, and even position elements on the page.
- CSS can be added directly to any HTML element (via the style attribute), within the <head> element, or, most commonly, in a separate text file that contains only CSS.

Benefits of CSS

The benefits of CSS include:

- **Improved control over formatting.** CSS gives fine-grained control over the appearance of their web content.
- **Improved site maintainability.** Websites become significantly more maintainable because all formatting can be centralized into one CSS file. This allows us to make site-wide visual modifications by changing a single file.
- **Improved accessibility.** CSS-driven sites are more accessible. By keeping presentation out of the HTML, screen readers and other accessibility tools work better, thereby providing a significantly enriched experience for those reliant on accessibility tools.
- **Improved page download speed.** A site built using a centralized set of CSS files for all presentation will also be quicker to download because each individual HTML file will contain less style information and markup, and thus be smaller.
- **Improved output flexibility.** CSS can be used to adopt a page for different output media.

CSS Versions

- Netscape's proposal was one that required the use of JavaScript programming to perform style changes.
- The W3C decided to adopt CSS, and by the end of 1996 the CSS Level 1 Recommendation was published. A year later, the CSS Level 2 Recommendation (also more succinctly labeled simply as CSS2) was published.
- A different group at the W3C was working on a CSS3 draft.
- To make CSS3 more manageable for both browser manufacturers and web designers, theW3C has subdivided it into a variety of different **CSS3 modules**.

Browser Adoption

- Microsoft's Internet Explorer was an early champion of CSS (its IE3, released in 1996, was the first major browser to support CSS, and its IE5 for the Macintosh was the first browser to reach almost 100% CSS1 support in 2000).

- Its later versions (especially IE5, IE6, and IE7) for Windows had uneven support for certain parts of CSS2. However, all browsers have not implemented parts of the CSS2 Recommendation.
- CSS has a reputation for being a somewhat frustrating language. Since CSS was designed to be a styling language, text styling is quite easy.

CSS Syntax

- A CSS document consists of one or more **style rules**.
- A rule consists of a **selector** that identifies the HTML element or elements that will be affected, followed by a series of **property:value pairs** (each pair is also called a **declaration**), as shown in Figure 2.1.
- The series of declarations is also called the **declaration block**. A declaration block can be together on a single line, or spread across multiple lines.
- The browser ignores white space (i.e., spaces, tabs, and returns) between our CSS rules so we can format the CSS however we want. Each declaration is terminated with a semicolon.
- The semicolon for the last declaration in a block is in fact optional. However, it is sensible practice to also terminate the last declaration with a semicolon as well.

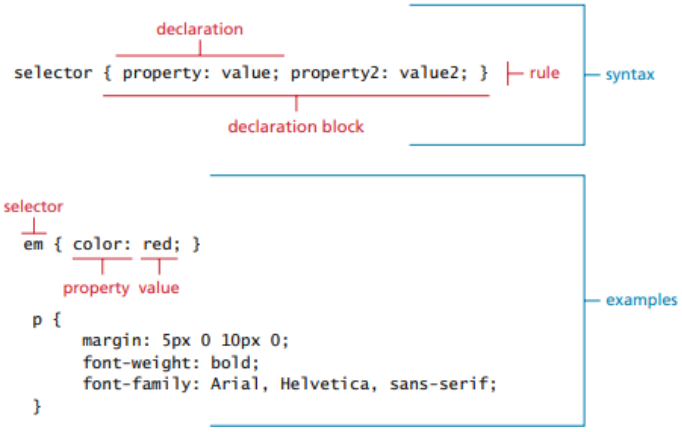


Figure 2.1: CSS Syntax

Selectors

- Every CSS rule begins with a **selector**.
- The selector identifies which element or elements in the HTML document will be affected by the declarations in the rule.

Properties

- Each individual CSS declaration must contain a property.
- These property names are predefined by the CSS standard.
- Table 2.1 lists many of the most commonly used CSS properties.

Property Type	Property
Fonts	font font-family font-size font-style font-weight @font-face
Text	letter-spacing line-height text-align text-decoration text-indent
Color and background	background background-color background-image background-position background-repeat color
Borders	border border-color border-width border-style border-top border-top-color border-top-width etc.
Spacing	padding padding-bottom, padding-left, padding-right, padding-top margin margin-bottom, margin-left, margin-right, margin-top
Sizing	height max-height max-width min-height min-width width
Layout	bottom, left, right, top clear display float overflow position visibility z-index
Lists	list-style list-style-image list-style-type

Table 2.1: Common CSS Properties

Values

- Each CSS declaration also contains a value for a property.
- The unit of any given value is dependent upon the property. Some property values are from a predefined list of keywords.
- Others are values such as length measurements, percentages, numbers without units, color values, and URLs.
- Colors would seem at first glance to be the most clear of these units.

Method	Description	Example
Name	Use one of 17 standard color names. CSS3 has 140 standard names.	`color: red;` `color: hotpink; /* CSS3 only */`
RGB	Uses three different numbers between 0 and 255 to describe the red, green, and blue values of the color.	`color: rgb(255,0,0);` `color: rgb(255,105,180);`
Hexadecimal	Uses a six-digit hexadecimal number to describe the red, green, and blue value of the color; each of the three RGB values is between 0 and FF (which is 255 in decimal). Notice that the hexadecimal number is preceded by a hash or pound symbol (#).	`color: #FF0000;` `color: #FF69B4;`
RGBa	This defines a partially transparent background color. The "a" stands for "alpha", which is a term used to identify a transparency that is a value between 0.0 (fully transparent) and 1.0 (fully opaque).	`color: rgb(255,0,0, 0.5);`
HSL	Allows you to specify a color using Hue Saturation and Light values. This is available only in CSS3. HSLA is also available as well.	`color: hsl(0,100%,100%);` `color: hsl(330,59%,100%);`

Table 2.2: Color Values

- Table 2.2 lists the different ways we can describe a color value in CSS just as there are multiple ways of specifying color in CSS, so there are multiple ways of specifying a unit of measurement.
- When working with print design, we generally make use of straightforward absolute units such as inches or centimeters and picas or points. Because different devices have differing physical sizes as well as different pixel resolutions and because the user is able to change the browser size or its zoom mode, these absolute units don't always make sense with web element measures.

Unit	Description	Type
px	Pixel. In CSS2 this is a relative measure, while in CSS3 it is absolute (1/96 of an inch).	Relative (CSS2) Absolute (CSS3)
em	Equal to the computed value of the font-size property of the element on which it is used. When used for font sizes, the em unit is in relation to the font size of the parent.	Relative
%	A measure that is always relative to another value. The precise meaning of % varies depending upon the property in which it is being used.	Relative
cm	Centimeters	Absolute
mm	Millimeters	Absolute
pt	Points (equal to 1/72 of an inch)	Absolute
Pc	Pica (equal to 1/6 of an inch)	Absolute

Unit	Description	Type
ex	A rarely used relative measure that expresses size in relation to the x-height of an element's font.	Relative
ch	Another rarely used relative measure; this one expresses size in relation to the width of the zero ("0") character of an element's font.	Relative (CSS3 only)
rem	Stands for root em, which is the font size of the root element. Unlike em, which may be different for each element, the rem is constant throughout the document.	Relative (CSS3 only)
vw, vh	Stands for viewport width and viewport height. Both are percentage values (between 0 and 100) of the viewport (browser window). This allows an item to change size when the viewport is resized.	Relative (CSS3 only)
in	Inches	Absolute

Table 2.3: Units of Measure Values

- Table 2.3 lists the different units of measure in CSS.
- **Relative units** are based on the value of something else or specifies a length relative to another length property, such as the size of a parent element.
- **Absolute units** have a real-world size or it is fixed in relation to each other.
- In general, most of the CSS that we will see uses either px, em, or % as a measure unit.

Location of Styles

- CSS style rules can be located in three different locations. These three are not mutually exclusive, in that we could place our style rules in all three.

1. Inline Styles

- **Inline styles** are style rules placed within an HTML element via the style attribute, as shown in Listing 2.1.
- An inline style only affects the element it is defined within and overrides any other style definitions for properties used in the inline style.
- A selector is not necessary with inline styles but semicolons are only required for separating multiple rules.

Advantages

- Inline styles are handy for quickly testing out a style change.
- We don't need to create and upload a separate document for css.

Disadvantages

- Adding CSS rules to every HTML element is time-consuming and make our HTML structure messy.
- Styling multiple elements can affect our page's size and download time.

```
<h1>Share Your Travels</h1>
<h2>style="font-size: 24pt"Description</h2>
...
<h2>style="font-size: 24pt; font-weight: bold;">Reviews</h2>
```

Listing 2.1: Internal styles example

2. Embedded Style Sheet

- Embedded style sheets (also called **internal styles**) are style rules placed within the<style> element (inside the <head> element of an HTML document), as shown in Listing 2.2.

Advantage

- Just as with inline styles, embedded styles can, however, be helpful when quickly testing out a style that is used in multiple places within a single HTML document.

Disadvantages

- While better than inline styles, using embedded styles is also by and large discouraged. Since each HTML document has its own <style> element, it is more difficult to consistently style multiple documents when using embedded styles.
- Adding the code to the HTML document can increase the page's size and loading time.

```
<head lang="en">
   <meta charset="utf-8">
   <title>Share Your Travels -- New York - Central Park</title>
   <style>
      h1 { font-size: 24pt; }
      h2 {
       font-size: 18pt;
       font-weight: bold;
       }
   </style>
</head>
<body>
   <h1>Share Your Travels</h1>
   <h2>New York - Central Park</h2>
   ...
```

Listing 2.2: Embedded styles example

3. External Style Sheet

- External style sheets are style rules placed within an external text file with the **.css** extension.
- To reference an external style sheet, you must use a <link> element (within the<head> element), as shown in Listing 2.3.

Advantages

- The most common place to locate style rules because it provides the best maintainability.
- When you make a change to an external style sheet, all HTML documents that reference that style sheet will automatically use the updated version.
- The browser is able to cache the external style sheet, which can improve the performance of the site as well.
- We can link to several style sheets at a time; each linked style sheet will require its own <link> element.

Disadvantage

- Pages may not be rendered correctly until the external CSS is loaded.
- Uploading or linking to multiple CSS files can increase our site's download time

```
<head lang="en">
   <meta charset="utf-8">
   <title>Share Your Travels -- New York - Central Park</title>
   <link rel="stylesheet" href="styles.css" />
</head>
```

Listing 2.3: Referencing an external style sheet

Selectors

- When defining CSS rules, we need to first use a selector to tell the browser which elements will be affected by the property values.
- CSS selectors allow us to select individual or multiple HTML elements.
- There are now a variety of new selectors that are supported by most modern browsers.

Element Selectors

- Element selectors select all instances of a given HTML element. The example CSS rules in Figure 2.1 illustrate two element selectors.
- We can select all elements by using the **universal element selector**, which is the * (asterisk) character.
- We can select a group of elements by separating the different element names with commas as shown in Listing 2.4.

```
/* commas allow you to group selectors */
p, div, aside {
   margin: 0;
   padding: 0;
}
/* the above single grouped selector is equivalent to the
   following: */
p {
   margin: 0;
   padding: 0;
}
div {
   margin: 0;
   padding: 0;
}
aside {
   margin: 0;
   padding: 0;
}
```

Listing 2.4: Element/grouped selector

Class Selectors

- A class selector allows us to simultaneously target different HTML elements regardless of their position in the document tree.
- If a series of HTML elements have been labeled with the same class attribute value, then we can target them for styling by using a class selector, which takes the form: period (.) followed by the class name.
- Listing 2.5 illustrates an example of styling using a class selector. The result in the browser is shown in Figure 2.2

```
<head>
   <title>Share Your Travels </title>
   <style>
          .first {
           font-style: italic;
           color: red;
          }
    </style>
</head>
<body>
   <h1 class="first">Reviews</h1>
   <div>
      <p class="first">By Ricardo on <time>September 15, 2015</time></p>
      <p>Easy on the HDR buddy.</p>
   </div>
   <hr/>

   <div>
      <p class="first">By Susan on <time>October 1, 2015</time></p>
      <p>I love Central Park.</p>
   </div>
   <hr/>
</body>
```

Listing 2.5: Class selector example

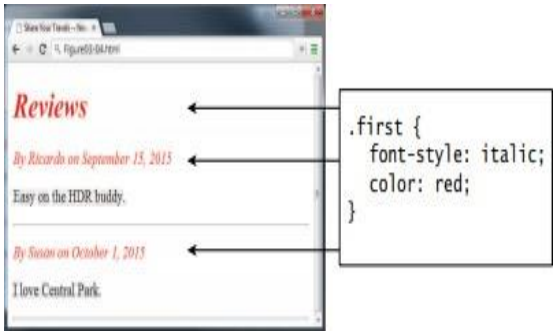


Figure 2.2: Class selector example in browser

Id Selectors

- An **id selector** allows us to target a specific element by its id attribute regardless of its type or position.
- If an HTML element has been labeled with an id attribute, then we can target it for styling by using an id selector, which takes the form: pound/hash (#) followed by the id name.
- Listing 2.6 illustrates an example of styling using an id selector. The result in the browser is shown in Figure2.3.

```
<head lang="en">
   <meta charset="utf-8">
   <title>Share Your Travels -- New York - Central Park</title>
   <style>
          #latestComment {
           font-style: italic;
           color: red;
          }
</style>
</head>
<body>
   <h1>Reviews</h1>
   <div id="latestComment">
      <p>By Ricardo on <time>September 15, 2015</time></p>
      <p>Easy on the HDR buddy.</p>
   </div>
   <hr/>
   <div>
      <p>By Susan on <time>October 1, 2015</time></p>
      <p>I love Central Park.</p>
   </div>
   <hr/>
</body>
```

Listing 2.6: Id selector example

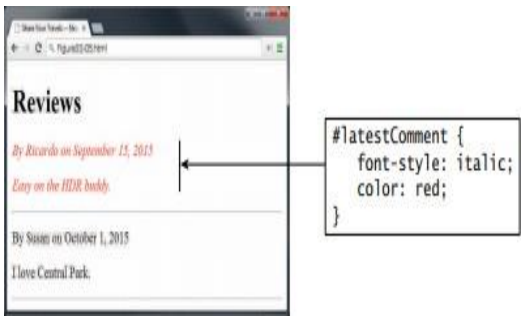


Figure 2.3: Id selector example in browser

Attribute Selectors

- An **attribute selector** provides a way to select HTML elements either by the presence of an element attribute or by the value of an attribute.

- Attribute selectors can be a very helpful technique in the styling of hyperlinks and images.
- We can do this by using the following attribute selector:

[title] { … }

- This will match any element in the document that has a title attribute.
- Table 2.4 summarizes some of the most common ways one can construct attribute selectors in CSS3. Listing 2.7, with the results in the browser shown in Figure 2.4.

```
<head lang="en">
   <meta charset="utf-8">
   <title>Share Your Travels</title>
     <style>
          [title] {
          cursor: help;
          padding-bottom: 3px;
          border-bottom: 2px dotted blue;
          text-decoration: none;
         }
   </style>
</head>
<body>
   <div>
      <img src="images/flags/CA.png" title="Canada Flag" />
      <h2><a href="countries.php?id=CA" title="see posts from Canada">
          Canada</a>
      </h2>
      <p>Canada is a North American country consisting of ... </p>
      <div>
          <img src="images/square/6114907897.jpg"
               title="At top of Sulphur Mountain" />
          <img src="images/square/6592317633.jpg"
               title="Grace Presbyterian Church" />
          <img src="images/square/6592914823.jpg"
               title="Calgary Downtown" />
     </div>
   </div>
</body>
```

Listing 2.7: Attribute selector example

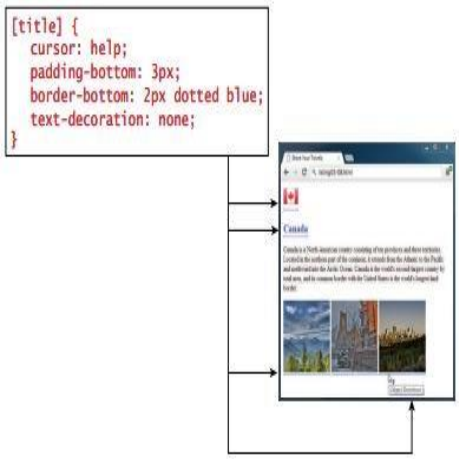


Figure 2.4: Attribute selector example in browser

Selector	Matches	Example
[]	A specific attribute.	`[title]` Matches any element with a title attribute
[=]	A specific attribute with a specific value.	`a[title="posts from this country"]` Matches any <a> element whose title attribute is exactly "posts from this country"
[~=]	A specific attribute whose value matches at least one of the words in a space-delimited list of words.	`[title~="Countries"]` Matches any `title` attribute that contains the word "Countries"
[^=]	A specific attribute whose value begins with a specified value.	`a[href^="mailto"]` Matches any <a> element whose `href` attribute begins with "mailto"
[*=]	A specific attribute whose value contains a substring.	`img[src*="flag"]` Matches any <img> element whose `src` attribute contains somewhere within it the text "flag"
[$=]	A specific attribute whose value ends with a specified value.	`a[href$=".pdf"]` Matches any <a> element whose `href` attribute ends with the text ".pdf"

Table 2.4: Attribute Selectors

Pseudo-Element and Pseudo-Class Selectors

- A **pseudo-element selector** is a way to select something that does not exist explicitly as an element in the HTML document tree but which is still a recognizable selectable object. For instance, we can select the first line or first letter of any HTML element using a pseudo-element selector.
- A **pseudo-class selector** does apply to an HTML element, but targets either a particular state or, in CSS3, a variety of family relationships.
- Table 2.5 lists some of the more common pseudo-class and pseudo element selectors. By default, the browser displays link text blue and visited text links purple.
- Listing 2.8 illustrates the use of pseudo-class selectors to style not only the visited and unvisited link colors, but also the hover color, which is the color of the link when the mouse is over the link.
- Note the syntax of pseudo-class selectors: the colon (:) followed by the pseudo-class selector name. Do be aware that a space is *not* allowed after the colon. The order of these pseudo-class elements is important.

Selector	Type	Description
`a:link`	pseudo-class	Selects links that have not been visited
`a:visited`	pseudo-class	Selects links that have been visited
`:focus`	pseudo-class	Selects elements (such as text boxes or list boxes) that have the input focus.
`:hover`	pseudo-class	Selects elements that the mouse pointer is currently above.
`:active`	pseudo-class	Selects an element that is being activated by the user. A typical example is a link that is being clicked.
`:checked`	pseudo-class	Selects a form element that is currently checked. A typical example might be a radio button or a check box.
`:first-child`	pseudo-class	Selects an element that is the first child of its parent. A common use is to provide different styling to the first element in a list.
`:first-letter`	pseudo-element	Selects the first letter of an element. Useful for adding drop-caps to a paragraph.
`:first-line`	pseudo-element	Selects the first line of an element.

Table 2.5: Common Pseudo-Class and Pseudo-Element Selectors

```
<head>
   <title>Share Your Travels</title>
  <style>
        a:link {
        text-decoration: underline;
        color: blue;
     }
        a:visited {
        text-decoration: underline;
        color: purple;
     }
        a:hover {
        text-decoration: none;
        font-weight: bold;
     }
        a:active {
        background-color: yellow;
     }
  </style>
</head>
<body>
      <p>Links are an important part of any web page. To learn more about
         links visit the <a href="#">W3C</a> website.</p>
   <nav>
     <ul>
        <li><a href="#">Canada</a></li>
        <li><a href="#">Germany</a></li>
        <li><a href="#">United States</a></li>
     </ul>
   </nav>
</body>
```

Listing 2.8: Styling a link using pseudo-class selectors

Contextual Selectors

- A **contextual selector** (in CSS3 also called **combinators**) allows you to select elements based on their *ancestors*, *descendants*, or *siblings*. That is, it selects elements based on their context or their relation to other elements in the document tree.
- While some of these contextual selectors are used relatively infrequently, almost all web authors find themselves using descendant selectors.
- A **descendant selector** matches all elements that are contained within another element. The character used to indicate descendant selection is the space character. Figure 2.5 illustrates the syntax and usage of the syntax of the descendant selector.
- Table 2.6 describes the other contextual selectors. Figure 2.6 illustrates some sample uses of a variety of different contextual selectors.

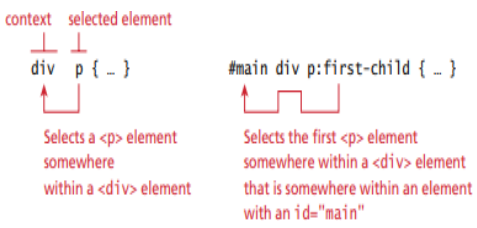


Figure 2.5: Syntax of a descendant selection

Selector	Matches	Example
Descendant	A specified element that is contained somewhere within another specified element.	`div p` Selects a <p> element that is contained somewhere within a element. That is, the <p> can be any descendant, not just a child.
Child	A specified element that is a direct child of the specified element.	`div>h2` Selects an <h2> element that is a child of a element.
Adjacent sibling	A specified element that is the next sibling (i.e., comes directly after) of the specified element.	`h3+p` Selects the first <p> after any <h3>.
General sibling	A specified element that shares the same parent as the specified element.	`h3~p` Selects all the <p> elements that share the same parent as the <h3>.

Table 2.6: Contextual Selectors

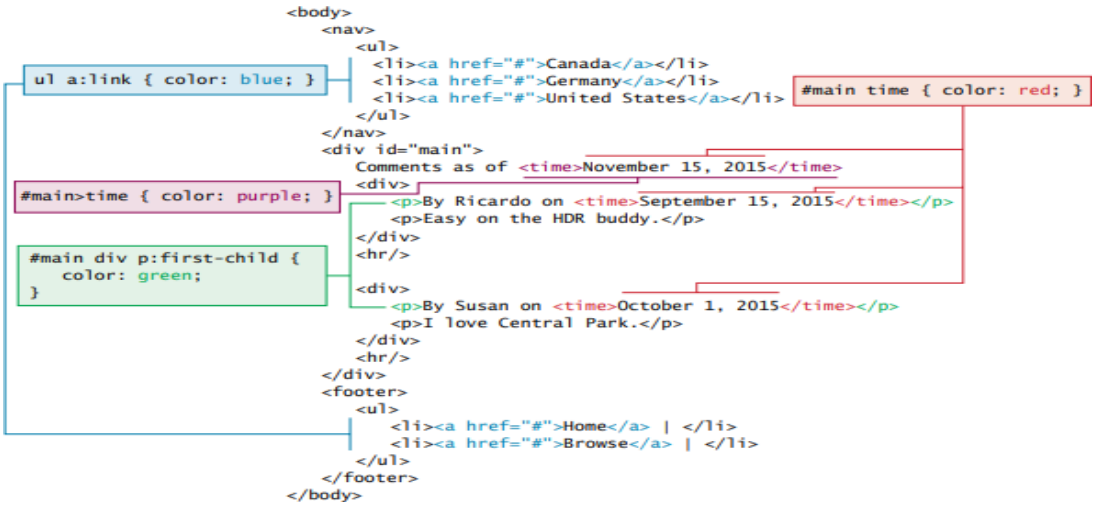


Figure 2.6: Contextual selectors in action

The Cascade: How Styles Interact

- There are three different types of style sheets: author-created, user-defined, and the default browser style sheet. As well, it is possible within an author-created style sheet to define multiple rules for the same HTML element
- CSS has a system to help the browser determine how to display elements when different style rules conflict. The "Cascade" in CSS refers to how conflicting rules are handled.
- CSS uses the following cascade principles to help it deal with conflicts: inheritance, specificity, and location.

Inheritance

- **Inheritance** is the first of these cascading principles. Many CSS properties affect not only themselves but their descendants as well.
    - Font, color, list, and text properties are **inheritable;**
    - layout, sizing, border, background, and spacing properties are **not inheritable.**
- Figures 2.7 illustrate CSS inheritance. In the first example, only some of the property rules are inherited for the <body> element. That is, only the body element (thankfully!) will have a thick green border and the 100-px margin.
- In the second example in Figure 2.8, we can assume there is no longer the body styling but instead we have a single style rule that styles *all* the elements. The <p> and <time> elements within the inherit the bold font-weight property but not the margin or border styles.

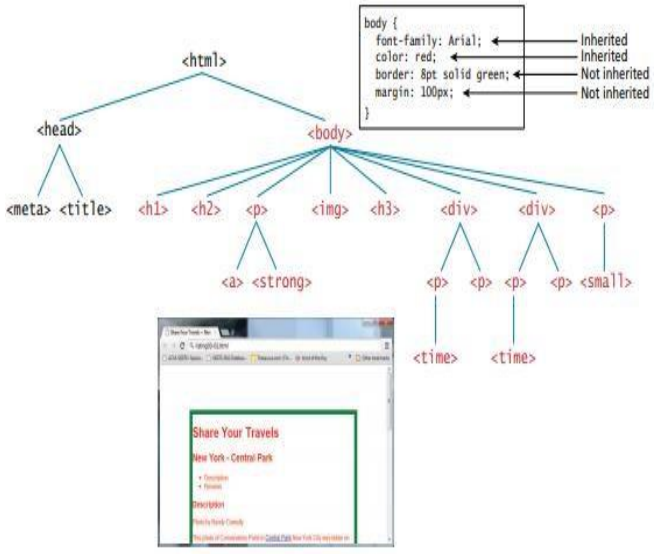


Figure 2.7: Inheritance

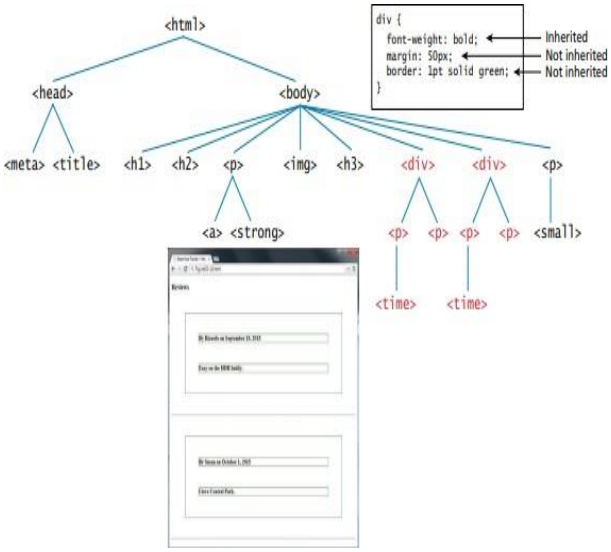


Figure 2.8: More Inheritance

- However, it is possible to tell elements to inherit properties that are normally not inheritable.

Specificity

- **Specificity** is how the browser determines which style rule takes precedence when more than one style rule could be applied to the same element.
- In CSS, the more specific the selector, the more it takes precedence (i.e., overrides the previous definition).
- Another way to define specificity is by telling us how it works. The way that specificity works in the browser is that the browser assigns a weight to each style rule; when several rules apply, the one with the greatest weight takes precedence.
- Figure 2.9, the color and font-weight properties defined in the <body> element are inheritable and thus potentially applicable to all the child elements contained within it.
- The and <p> elements also have the same properties set, they *override* the value defined for the<body> element because their selectors (and <p>) are more specific.
- Figure 2.9, class selectors take precedence over element selectors, and id selectors take precedence over class selectors.

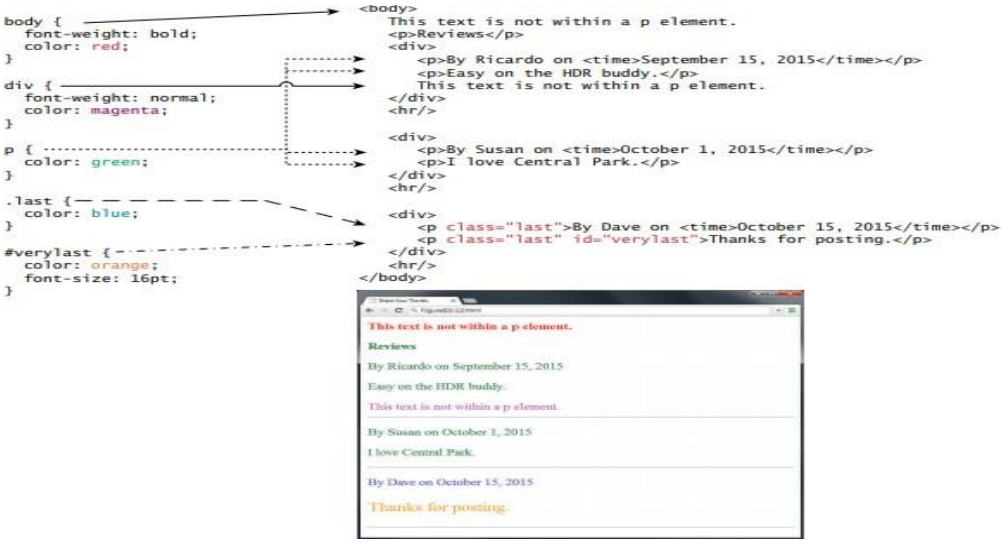


Figure 2.9: Specificity

Location

- When inheritance and specificity cannot determine style precedence, the principle of **location** will be used.
- The principle of location is that when rules have the same specificity, then the latest are given more weight. For instance, an inline style will override one defined in an external author style sheet or an embedded style sheet.
- Similarly, an embedded style will override an equally specific rule defined in an external author style sheet if it appears after the external sheet's <link> element.
- Similarly, when the same style property is defined multiple times within a single declaration block, the last one will take precedence as shown in Figure 2.10.

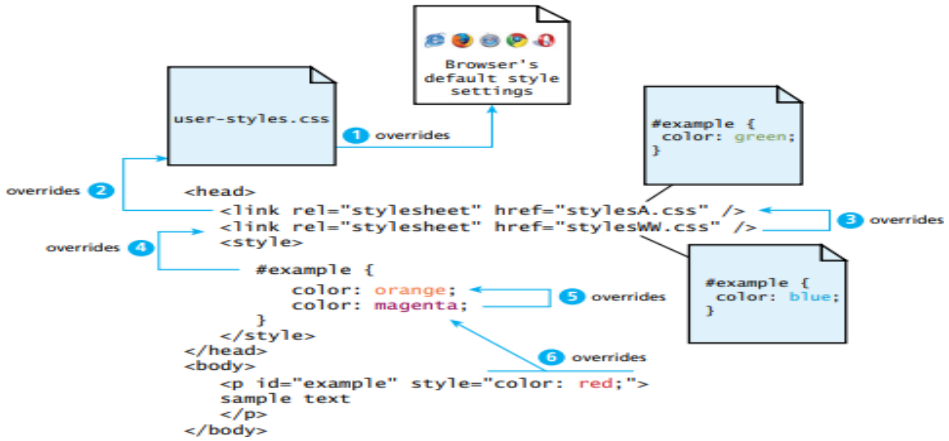


Figure 2.10: Location

The Box Model

In CSS, all HTML elements exist within an **element box** shown in Figure 2.11.

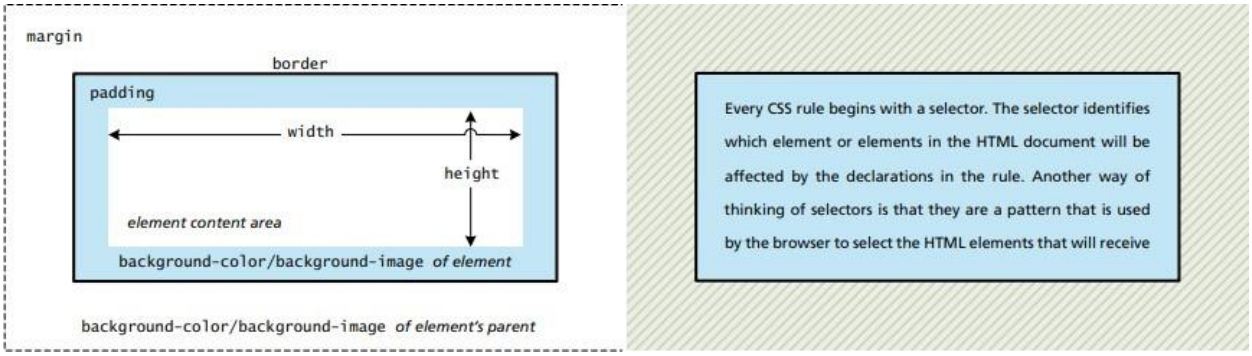


Figure 2.11: CSS box model

Background

- The background color or image of an element fills an element out to its border.
- In contemporary web design, it has become extremely common to use CSS to display purely presentational images (such as background gradients and patterns, decorative images, etc.) rather than using the <img> element.
- Table 2.7 lists the most common background properties

Property	Description
background	A combined shorthand property that allows you to set multiple background values in one property. While you can omit properties with the shorthand, do remember that any omitted properties will be set to their default value.
background-attachment	Specifies whether the background image scrolls with the document (default) or remains fixed. Possible values are: fixed, scroll.
background-color	Sets the background color of the element. You can use any of the techniques shown in Table 3.2 for specifying the color.
background-image	Specifies the background image (which is generally a jpeg, gif, or png file) for the element. Note that the URL is relative to the CSS file and not the HTML. CSS3 introduced the ability to specify multiple background images.
background-position	Specifies where on the element the background image will be placed. Some possible values include: bottom, center, left, and right. You can also supply a pixel or percentage numeric position value as well. When supplying a numeric value, you must supply a horizontal/vertical pair; this value indicates its distance from the top left corner of the element, as shown in Figure 3.16.
background-repeat	Determines whether the background image will be repeated. This is a common technique for creating a tiled background (it is in fact the default behavior), as shown in Figure 3.17. Possible values are: repeat, repeat-x, repeat-y, and no-repeat.
background-size	New to CSS3, this property lets you modify the size of the background image.

Table 2.7: Common Background Properties

Borders

- Borders provide a way to visually separate elements. We can put borders around all four sides of an element, or just one, two, or three of the sides. Table 2.8 lists the various border properties.

Property	Description
border	A combined shorthand property that allows you to set the style, width, and color of a border in one property. The order is important and must be: `border-style border-width border-color`
border-style	Specifies the line type of the border. Possible values are: `solid, dotted, dashed, double, groove, ridge, inset, and outset.`
border-width	The width of the border in a unit (but not percents). A variety of keywords (`thin`, `medium`, etc.) are also supported.
border-color	The color of the border in a color unit.
border-radius	The radius of a rounded corner.
border-image	The URL of an image to use as a border.

Table 2.8: Border Properties

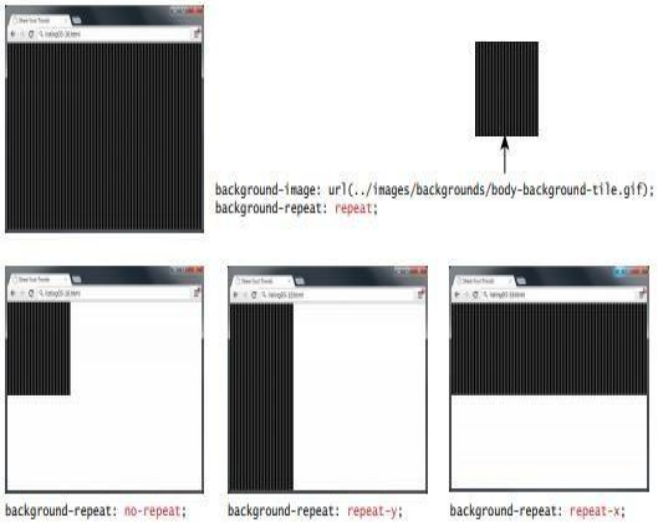


Figure 2.12: Background repeat

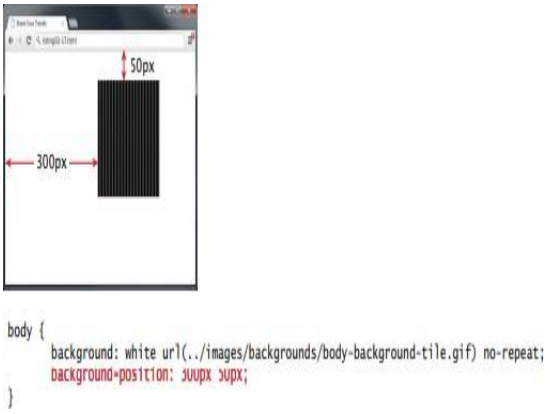


Figure 2.13: Background position

Margins and Padding

- Margins and padding are essential properties for adding white space to a web page, which can help differentiate one element from another.
- Figure 2.14 illustrates how these two properties can be used to provide spacing and element differentiation.
  - **Margins** add spacing around an element's content,

- while **padding** adds spacing within elements.

- Borders divide the margin area from the padding area.

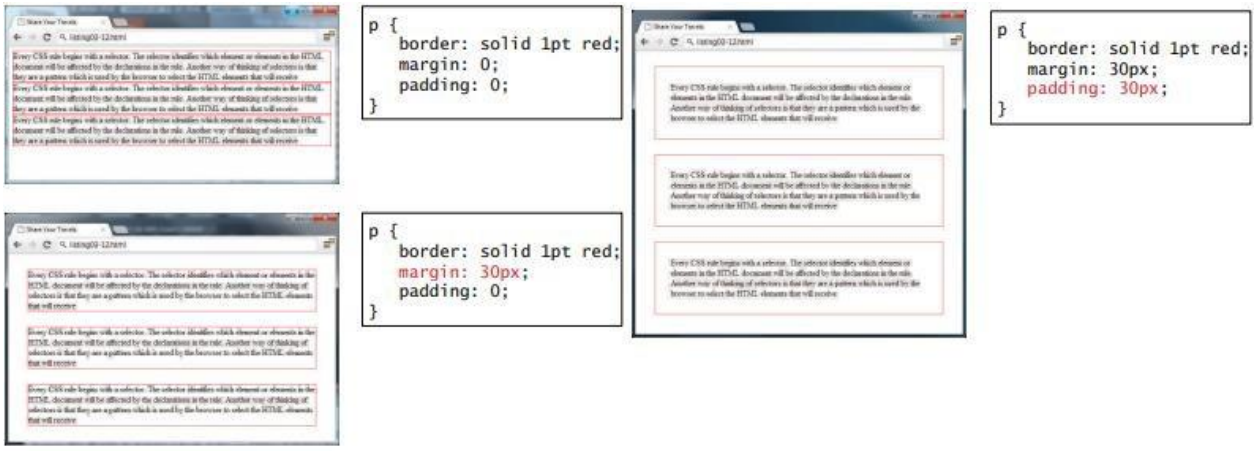


Figure 2.14: Borders, margins, and padding provide element spacing and differentiation

- Figure 2.15 illustrates how adjoining vertical margins collapse in the browser.
- If overlapping margins did not collapse, then margin space for 2 would be 180 px (90 px for the bottom margin of the first + 90 px for the top margin of the second), while the margins for 4 and 5 would be 100 px.
- The W3C specification defines this behavior as **collapsing margins**:
  - In CSS, the adjoining margins of two or more boxes (which might or might not be siblings) can combine to form a single margin. Margins that combine this way are said to collapse, and the resulting combined margin is called a collapsed margin.

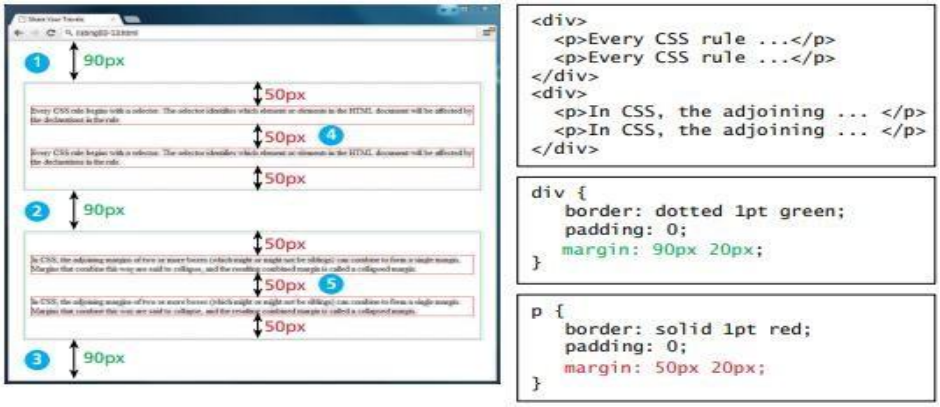


Figure 2.15: Collapsing vertical margins

- The **vertical** margins of two elements touch, only the largest margin value of the elements will be displayed, while the smaller margin value will be collapsed to zero.

- Horizontal margins, on the other hand, **never** collapse. To complicate matters even further, there are a large number of special cases in which adjoining vertical margins do **not** collapse.

Box Dimensions

- By default (in CSS this is the auto value), the width of and height of elements is the actual size of the content. For text, this is determined by the font size and font face; for images, the width and height of the actual image in pixels.
- Since the width and the height only refer to the size of the content area, the total size of an element is equal to the size of its content area plus the sum of its padding, borders, and margins.

height + padding + border = actual height of an element

width + padding + border = actual width of an element

- Figure 2.16 illustrates the default content-box element sizing behavior.

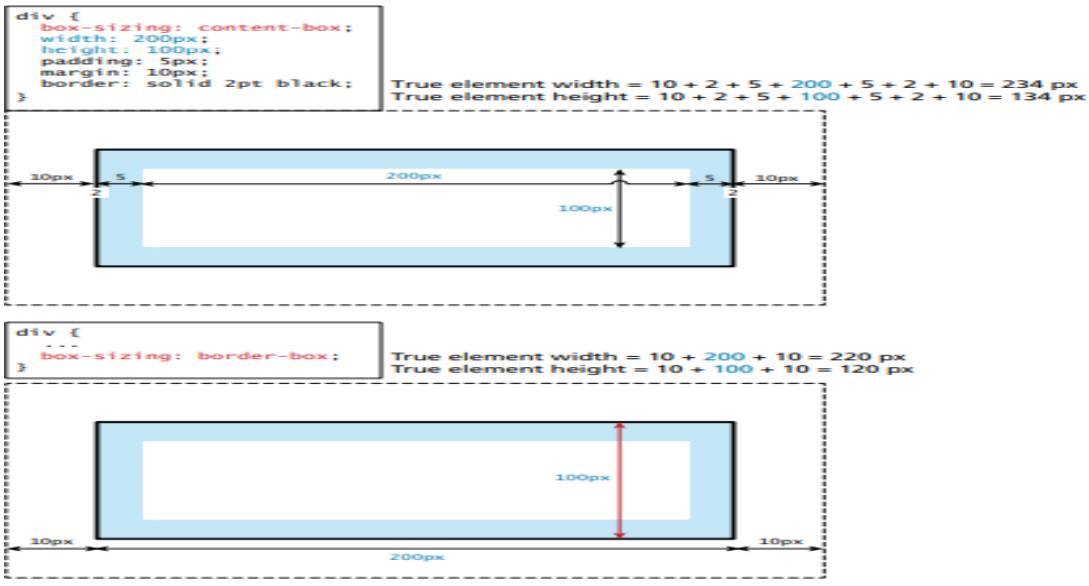


Figure 2.16: Calculating an element's true size

- For block-level elements such as <p> and elements, there are limits to what the width and height properties can actually do.
- We can shrink the width, but the content still needs to be displayed, so the browser may very well ignore the height that we set.

- Figure 2.17, the default width is the browser viewport. But in the second screen capture in the image, with the changed width and height, there is not enough space for the browser to display all the content with in the element. The height of the actual textual content is much larger (depending on the font size).

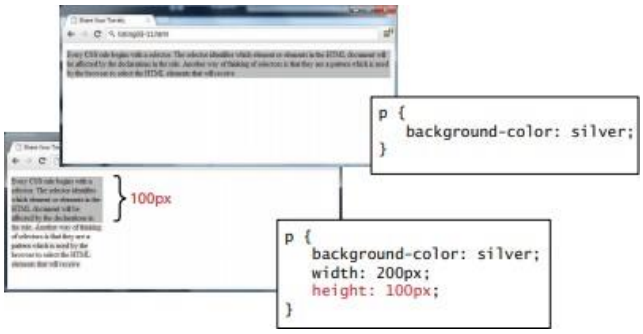


Figure 2.17: Limitations of height property

- It is possible to control the content if the box's width and height are not large enough to display the content using the overflow property, as shown in Figure 2.18.

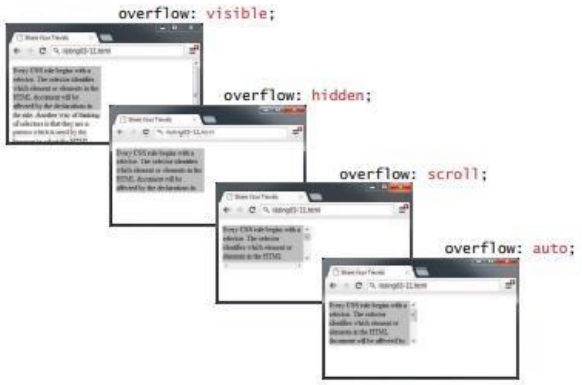


Figure 2.18: overflow property

- One of the problems with using percentages as the unit for sizes is that as the browser window shrinks too small or expands too large (for instance on a wide screen monitor), elements might become too small or too large. We can put absolute pixel constraints on the minimum and maximum sizes via the min-width, min-height, max-width, and max-height properties.

CSS Text Styling

- CSS provides two types of properties that affect text.
  - The first we call font properties because they affect the font and its appearance.

- The second type of CSS text properties are referred to here as paragraph properties since they affect the text in a similar way no matter which font is being used.

Font Family

- A word processor on a desktop machine can make use of any font that is installed on the computer; browsers are no different.
- It is conventional to supply a so-called web font stack, that is, a series of alternate fonts to use in case the original font choice is not on the user's computer.
- Figure 2.19, the alternatives are separated by commas; as well, if the font name has multiple words, then the entire name must be enclosed in quotes.

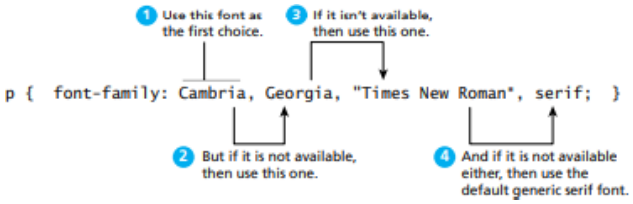


Figure 2.19: Specifying the font family

Property	Description
font	A combined shorthand property that allows you to set the family, style, size, variant, and weight in one property. While you do not have to specify each property, you must include at a minimum the font size and font family. In addtion, the order is important and must be: `style weight variant size font-family`
font-family	Specifies the typeface/font (or generic font family) to use. More than one can be specified.
font-size	The size of the font in one of the measurement units.
font-style	Specifies whether `italic`, `oblique` (i.e., skewed by the browser rather than a true italic), or `normal`.
font-variant	Specifies either `small-caps` text or none (i.e., regular text).
font-weight	Specifies either `normal`, `bold`, `bolder`, `lighter`, or a value between 100 and 900 in multiples of 100, where larger number represents weightier (i.e., bolder) text.

Table 2.9: font properties

- The font-family property supports five different generic families; the browser supports a type face from each family. The different generic font families are shown in Figure 2.20.

Figure 2.20: The different font families

Font Sizes

- One of the principles of the web is that the user should be able to change the size of the text if he or she so wishes to do so; using percentages or em units ensures that this user action will "work," and not break the page layout.
- When used to specify a font size, both em units and percentages are relative to the parent's font size. This takes some getting used to.
- Figure 2.21 illustrates a common set of percentages and their em equivalents to scale elements relative to the default 16-px font size

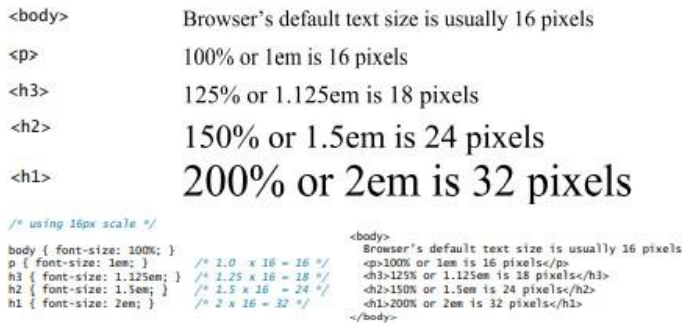


Figure 2.21: Using percents and em units for font sizes

- percents and em units are relative to their parents. Figure 2.22 illustrates how in reality it can quickly become difficult to calculate actual sizes when there are nested elements.

```
<body>
  <p>this is 16 pixels</p>
  <h1>this is 32 pixels</h1>
  <article>
     <h1>this is 32 pixels</h1>
     <p>this is 16 pixels</p>
     <div>
        <h1>this is 32 pixels</h1>
        <p>this is 16 pixels</p>
     </div>
  </article>
</body>
```


```
/* using 16px scale */

body { font-size: 100%; }
p    { font-size: 1em; }       /* 1 x 16 = 16px */
h1   { font-size: 2em; }       /* 2 x 16 = 32px */
```


```
/* using 16px scale */

body { font-size: 100%; }
p    { font-size: 1em; }
h1   { font-size: 2em; }

article { font-size: 75% }   /* h1 = 2 * 16 * 0.75 = 24px
                                p = 1 * 16 * 0.75 = 12px */

div  { font-size: 75% }      /* h1 = 2 * 16 * 0.75 * 0.75 = 18px
                                p = 1 * 16 * 0.75 * 0.75 = 9px */
```

Figure 2.22: Complications in calculating percents and em units

- To avoid complications, CSS3 supports a new relative measure, the rem (for root em unit). This unit is always relative to the size of the root element (i.e., the element).


```
/* using 16px scale */

body { font-size: 100%; }
p {
     font-size: 16px;  /* for older browsers: won't scale properly though */
     font-size: 1rem;  /* for new browsers: scales and simple too */
}
h1   { font-size: 2em; }

article { font-size: 75% }  /* h1 = 2 * 16 * 0.75 = 24px
                               p = 1 * 16 = 16px */

div  { font-size: 75% }     /* h1 = 2 * 16 * 0.75 * 0.75 = 18px
                               p = 1 * 16 = 16px */
```

Figure 2.23: Using rem units

Paragraph Properties

- Just as there are properties that affect the font in CSS, there are also a range of CSS properties that affect text independently of the font.
- Many of the most common text properties are shown in Table 2.10, and like the earlier font properties, many of these will be familiar to anyone who has used a word processor

Property	Description
letter-spacing	Adjusts the space between letters. Can be the value normal or a length unit.
line-height	Specifies the space between baselines (equivalent to leading in a desktop publishing program). The default value is normal, but can be set to any length unit. Can also be set via the shorthand font property.
list-style-image	Specifies the URL of an image to use as the marker for unordered lists.
list-style-type	Selects the marker type to use for ordered and unordered lists. Often set to none to remove markers when the list is a navigational menu or a input form.
text-align	Aligns the text horizontally in a container element in a similar way as a word processor. Possible values are left, right, center, and justify.
text-decoration	Specifies whether the text will have lines below, through, or over it. Possible values are: none, underline, overline, line-through, and blink. Hyperlinks by default have this property set to underline.
text-direction	Specifies the direction of the text, left-to-right (ltr) or right-to-left (rtl).
text-indent	Indents the first line of a paragraph by a specific amount.
text-shadow	A new CSS3 property that can be used to add a drop shadow to a text. Not yet supported in IE9.
text-transform	Changes the capitalization of text. Possible values are none, capitalize, lowercase, and uppercase.
vertical-align	Aligns the text vertically in a container element. Most common values are: top, bottom, and middle.
word-spacing	Adjusts the space between words. Can be the value normal or a length unit.

Table 2.10: Text Properties

**********End of Module_1**********

MODULE 2

Chapter 1: HTML Tables and Forms

1. Introducing Tables
2. Styling Tables
3. Introducing Forms
4. Form Control Elements
5. Table and form Accessibility
6. Microformats

Introducing Tables

- A **table** in HTML is created using the <table> element and can be used to represent information that exists in a two-dimensional grid.
- Tables can be used to display calendars, financial data, pricing tables, and many other types of data.
- HTML table can contain any type of data: like numbers, text, images, forms, even other tables.

Basic Table Structure

- HTML <table> contains any number of rows(<tr>); each row contains any number of table data cells (<td>) as shown in Figure 1.1.
- Some browsers do not display borders for the table by default; however, we can do so via CSS. Many tables will contain some type of headings in the first row.
- In HTML, we indicate header data by using the <th> instead of the <td> element as shown in Figure 1.2, because browsers tend to make the content within a <th> element bold, but this can also be done by CSS.
- The main reason we shouldn't use the <th> element is, due to presentation reasons. Rather, we should also use the <th> element for accessibility reasons and for search engine optimization reasons.

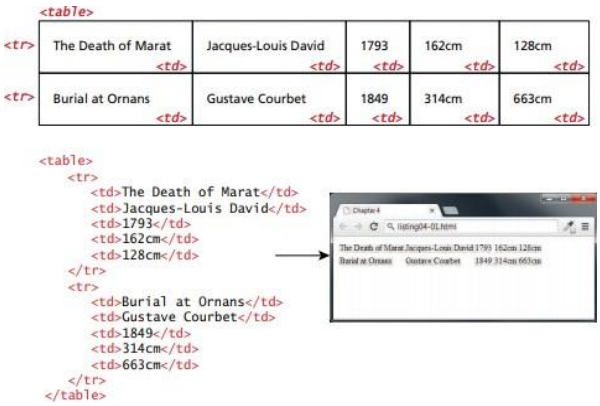


Figure 1.1: Basic Table Structure

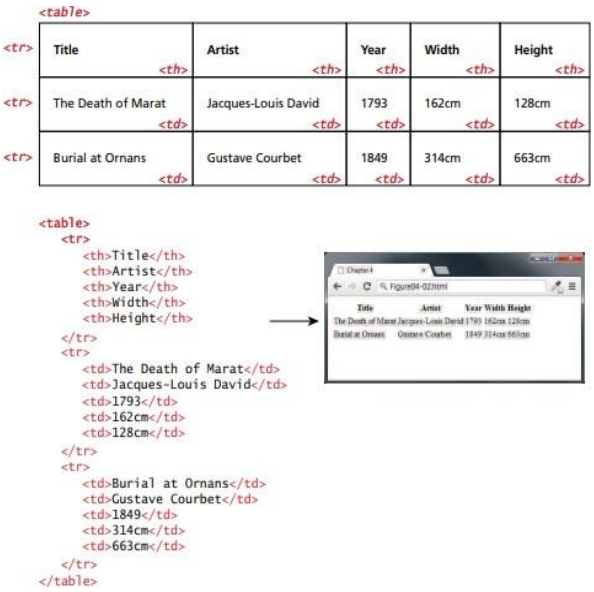


Figure 1.2: Adding table heading

Spanning Rows and Columns

- Two key features about tables are:
  1. The first is that all content must appear within the <td> or <th> container.
  2. The second is that each row must have the same number of <td> or <th> containers.
- There is a way to change this second behavior/feature.
  - i.e., If we want a given cell to cover several columns or rows, then we can do so by using the **colspan** (Figure 1.3) or **rowspan** attributes (Figure 1.4).

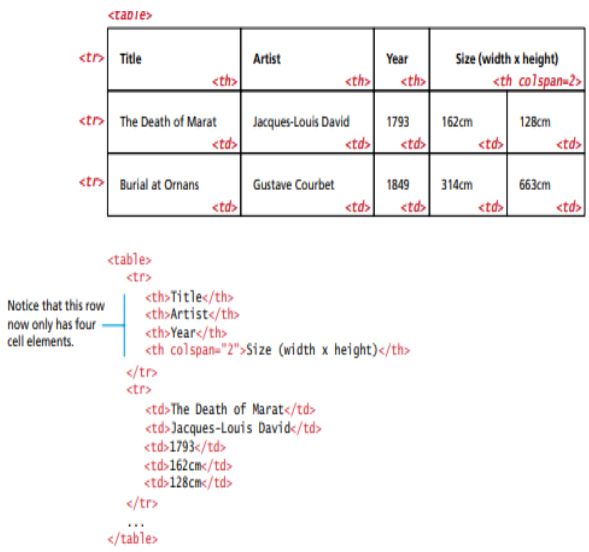


Figure 1.3: Spanning Columns

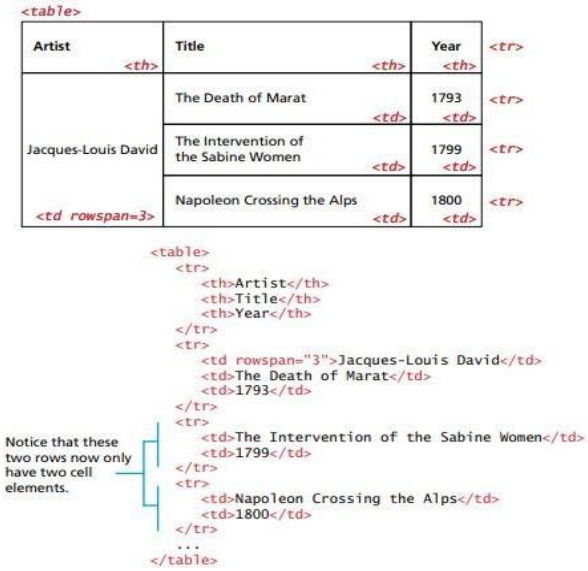


Figure 1.4: Spanning rows

Additional Table Elements

- The <caption> element is used to provide a brief title or description of the table, which improves the accessibility of the table, and is strongly recommended.
- We can use the caption-side CSS property to change the position of the caption below the table.
- The <thead>, <tfoot>, and <tbody> elements tend in practice to be used quite infrequently but make some sense for tables with a large number of rows.
- With CSS, one could set the height and overflow properties of the <tbody>element so that its content scrolls, while the header and footer of the table remain always on screen.
- The <col> and <colgroup> elements are also mainly used to aid in the eventual styling of the table. Rather than styling each column, we can style all columns within a <colgroup> with just a single style.
- The only properties we can set via these two elements are borders, backgrounds, width, and visibility, and only if they are not overridden in a <td>, <th>, or <tr> element.

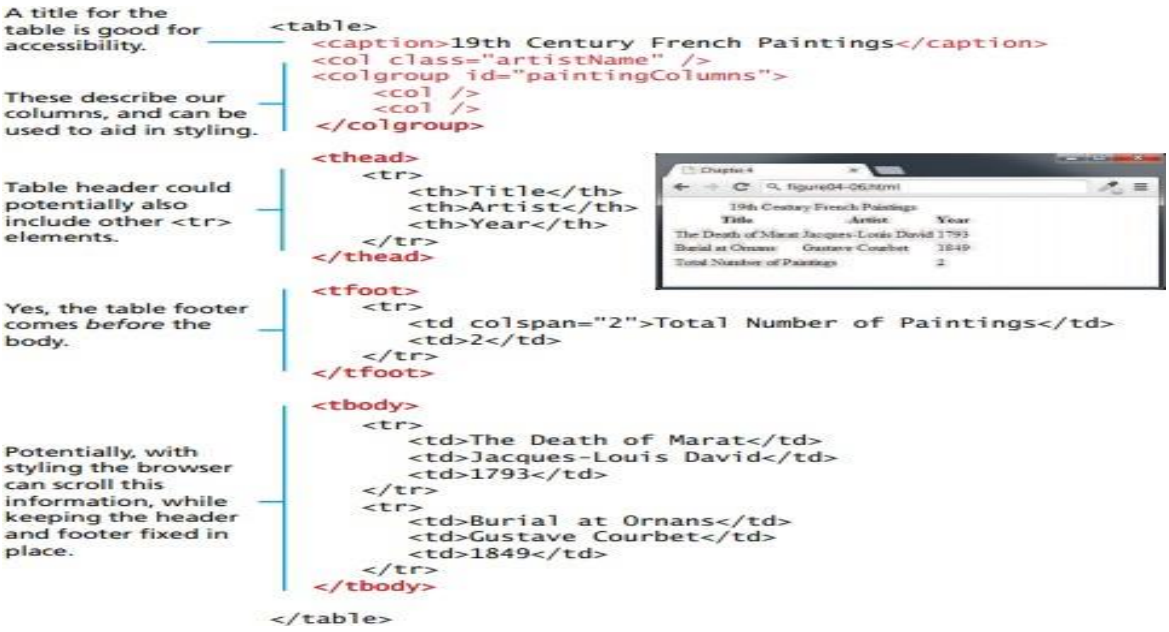


Figure 1.5: Additional table elements

Using Tables for Layout

- HTML tables were frequently used to create page layouts. Since HTML block-level elements exist on their own line, tables were embraced by developers in the 1990s as a way to get block-level HTML elements to sit side by side on the same line as shown in Figure 1.6.

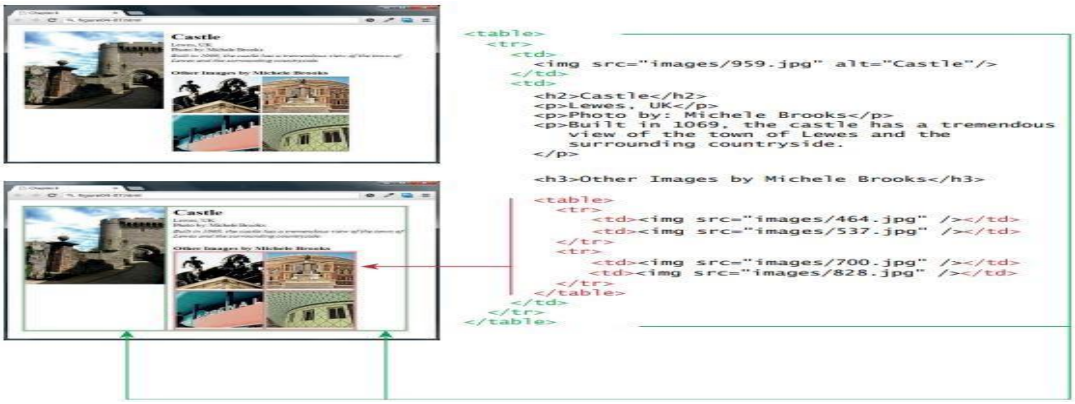


Figure 1.6: Example of using tables for layout

Problems:

1. It dramatically increase the size of the HTML document, so takes longer time to download and maintainability is difficult as it has many table elements.
2. The resulting markup is not semantic because tables are meant to indicate the tabular data but if we use table elements to align the block-elements side by side means we are giving presentation rather than semantic.
3. Using tables for layout results in a page that is not accessible, meaning that for users who rely on software to voice the content, table-based layouts can be extremely uncomfortable and confusing to understand. It is much better to use CSS for layout.

Styling Tables

- There is certainly no limit to the way one can style a table.

Table Borders

- Borders can be assigned to the <table>, <th> and the <td> element. Interestingly, borders cannot be assigned to the <tr>, <thead>, <tfoot>, and<tbody> elements.
- This property selects the table's border model. The default, shown in the second screen capture in Figure 1.7.
- In this approach, each cell has its own unique borders. We can adjust the space between these adjacent borders via the border-spacing property, as shown in the final screen capture in Figure 1.7.
- In the third screen capture, the collapsed border model is being used; in this model adjacent cells share a single border.

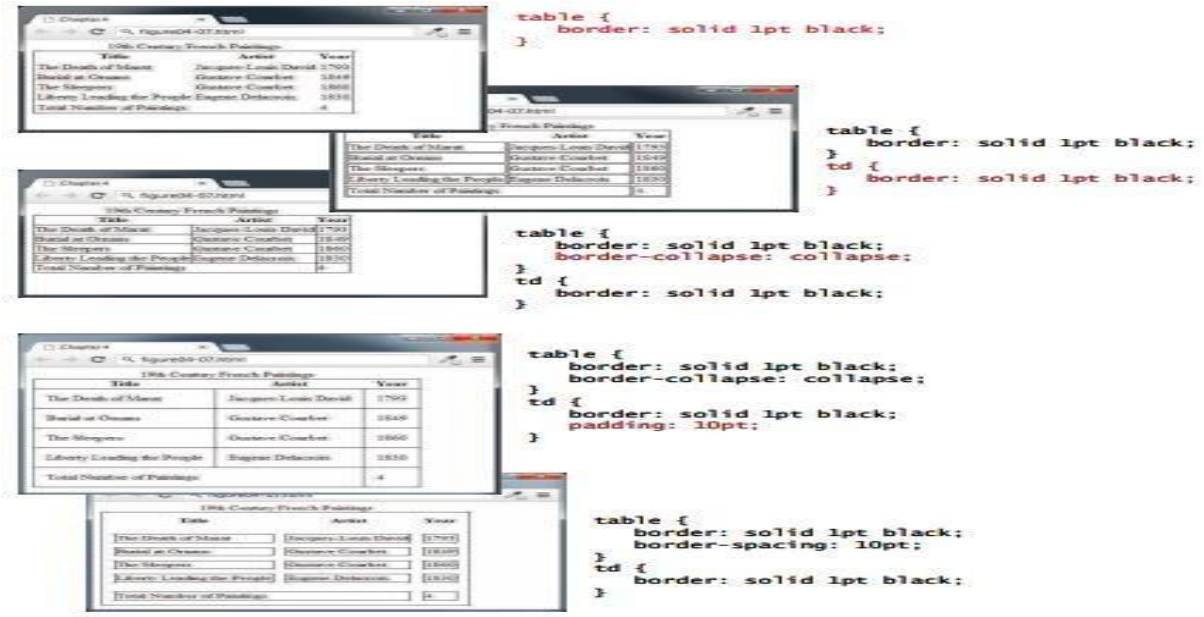


Figure 1.7: Styling table borders

Boxes and Zebras

- There are different ways to style a table.
- The first of these is a box format, in which we simply apply background colors and borders in various ways, as shown in Figure 1.8.
- We can then add special styling to the hover pseudo-class of the <tr> element, to highlight a row when the mouse cursor hovers over a cell, as shown in Figure 1.9.

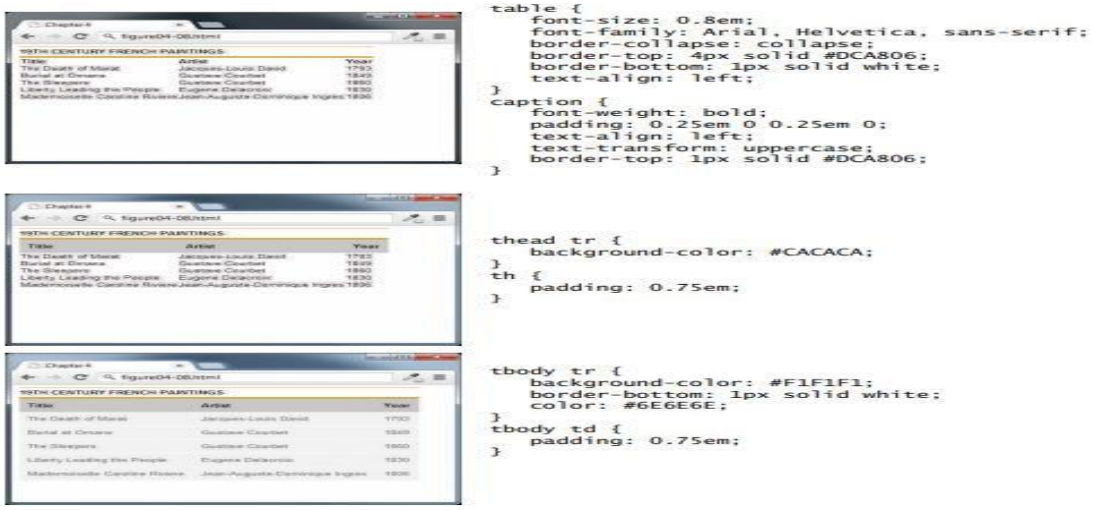
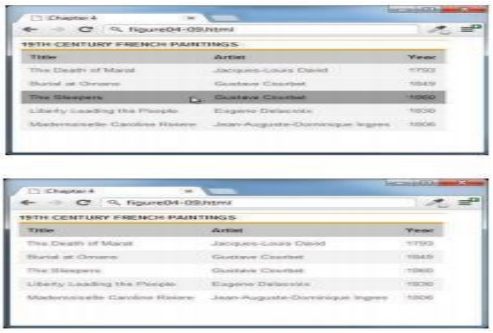


Figure 1.8: An example boxed table

- Figure 1.9 also illustrates how the pseudo-element nth-child can be used to alternate the format of every second row.



```
tbody tr:hover {
    background-color: #9e9e9e;
    color: black;
}
```



```
tbody tr:nth-child(odd) {
    background-color: white;
}
```

Figure 1.9: Hover effect and zebra stripes

Introducing Forms

- **Forms** provide an alternative way to interact with a web server. Forms provide a much richer mechanism.
- Using a form, the user can enter text, choose items from lists, and click buttons. Typically, programs running on the server will take the input from HTML forms and do something with subsequent HTML based on that input.
- There were controls for entering text, controls for choosing from a list, buttons, checkboxes, and radio buttons.
- HTML5 has added a number of new controls as well as more customization options for the existing controls.

Form Structure

- A form is constructed in HTML in the same manner as tables or lists: i.e., using special HTML elements (<form>).
- Figure 1.10 illustrates a typical HTML form. The form is defined by a <form> element, which is a container for other elements that represent the various input elements within the form as well as plain text and almost any other HTML element.

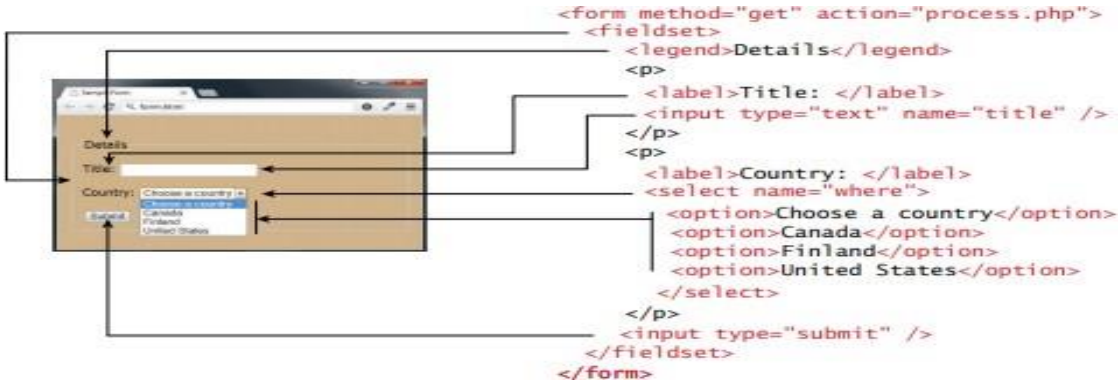


Figure 1.10: Simple HTML form

How Forms Work

- While forms are constructed with HTML elements, a form also requires some type of server-side resource that processes the user's form input as shown in Figure 1.11.
- The process begins with a request for an HTML page that contains some type of form on it. This could be something as complex as a user registration form or as simple as a search box.
- After the user fills out the form, there needs to be some mechanism for submitting the form data back to the server. This is achieved via a submit button, but through JavaScript, it is possible to submit form data using some other type of mechanism.
- Because interaction between the browser and the web server is governed by the HTTP protocol, the form data must be sent to the server via a standard HTTPrequest.

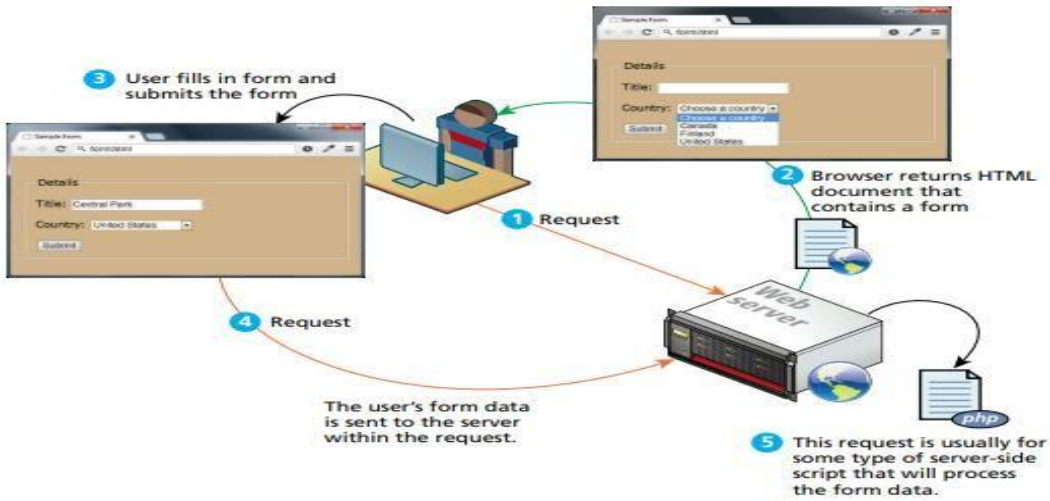


Figure 1.11: how forms work

Query Strings

- A **query string** is a series of name=value pairs separated by ampersands (the **&**character).
- Each form element (i.e., the first<input> elements and the <select> element) contains a name attribute, which is used to define the name for the form data in the query string.
- The values in the query string are the data entered by the user. Figure 1.12 illustrates how the form data (and its connection to form elements) is packaged into a query string.
- Query strings have certain rules defined by the HTTP protocol. Certain characters such as spaces, punctuation symbols, and foreign characters cannot be part of a query string.

- Instead, such special symbols must be **URL encoded** (also called **percent encoded**), as shown in Figure 1.13.

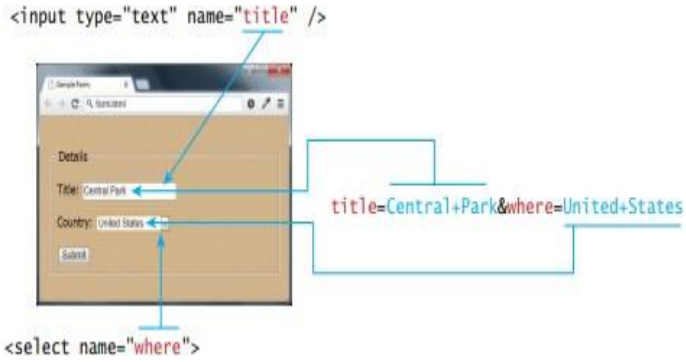


Figure 1.12: Query string data and its connection

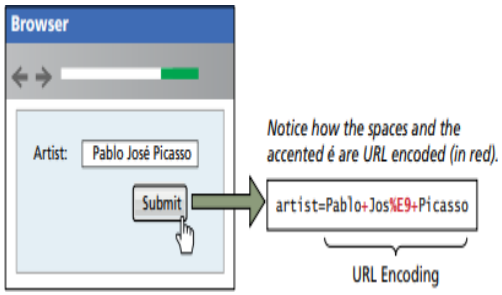


Figure 1.13: URL Encoding

The <form> Element

- Two important attributes that are essential features of any form, namely the action and the method attributes.
- The **action attribute** specifies the URL of the server-side resource that will process the form data. This could be a resource on the same server as the form or a completely different server.
- We will be using PHP pages to process the form data. There are other server technologies, each with their own extensions, such as ASP.NET (**.aspx**), ASP (**.asp**), and JavaServer Pages (**.jsp**).
- The **method attribute** specifies how the query string data will be transmitted from the browser to the server.
- There are two possibilities: GET and POST. The difference GET and POST resides in where the browser locates the user's form input in the subsequent HTTP request.
- With **GET**, the browser locates the data in the URL of the request; with **POST**, the form data is located in the HTTP header after the HTTP variables.
- Figure 1.14 illustrate show the two methods differ. Table 1.1 lists the key advantages and disadvantages of each method. Generally, form data is sent using the POST method.
- The GET method is useful when we are testing or developing a system, since you can examine the query string directly in the browser's address bar. Since the GET method uses the URL to transmit the query string,

- Form data will be saved when the user bookmarks a page, which may be desirable, but is generally a potential security risk. And needless to say, any time passwords are being transmitted, they should be transmitted via the POST method.

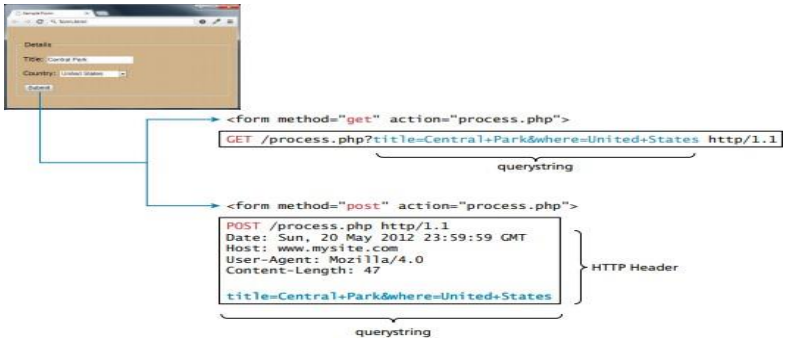


Figure 1.14: GET versus POST

Type	Advantages and Disadvantages
GET	Data can be clearly seen in the address bar. This may be an advantage during development but a disadvantage in production. Data remains in browser history and cache. Again this may be beneficial to some users, but a security risk on public computers. Data can be bookmarked (also an advantage and a disadvantage). Limit on the number of characters in the form data returned.
POST	Data can contain binary data. Data is hidden from user. Submitted data is not stored in cache, history, or bookmarks.

Table 1.1: GET versus POST

Type	Description
<button>	Defines a clickable button.
<datalist>	An HTML5 element that defines lists of pre-defined values to use with input fields.
<fieldset>	Groups related elements in a form together.
<form>	Defines the form container.
<input>	Defines an input field. HTML5 defines over 20 different types of input.
<label>	Defines a label for a form input element.
<legend>	Defines the label for a fieldset group.
<option>	Defines an option in a multi-item list.
<optgroup>	Defines a group of related options in a multi-item list.
<select>	Defines a multi-item list.
<textarea>	Defines a multiline text entry box.

Table 1.2: Form-Related HTML Elements

Form Control Elements

- Despite the wide range of different form input types in HTML5, there are only a relatively small number of form-related HTML elements, as shown in Table 1.2.

Text Input Controls

- Most forms need to gather text information from the user.
- Whether it is a search box, or a login form, or a user registration form, some type of text input is usually necessary.

Type	Description
text	Creates a single-line text entry box. <input type="text" name="title" />
textarea	Creates a multiline text entry box. You can add content text or if using an HTML5 browser, placeholder text (hint text that disappears once user begins typing into the field). <textarea rows="3" ... />
password	Creates a single-line text entry box for a password (which masks the user entry as bullets or some other character) <input type="password" ... />
search	Creates a single-line text entry box suitable for a search string. This is an HTML5 element. Some browsers on some platforms will style search elements differently or will provide a clear field icon within the text box. <input type="search" ... />
email	Creates a single-line text entry box suitable for entering an email address. This is an HTML5 element. Some devices (such as the iPhone) will provide a specialized keyboard for this element. Some browsers will perform validation when form is submitted. <input type="email" ... />
tel	Creates a single-line text entry box suitable for entering a telephone. This is an HTML5 element. Since telephone numbers have different formats in different parts of the world, current browsers do not perform any special formatting or validation. Some devices may, however, provide a specialized keyboard for this element. <input type="tel" ... />
url	Creates a single-line text entry box suitable for entering a URL. This is an HTML5 element. Some devices may provide a specialized keyboard for this element. Some browsers also perform validation on submission. <input type="url" ... />

Table 1.3: Text Input Controls

- Figure 1.15 illustrates the various text element controls and some examples of how they look in selected browsers.

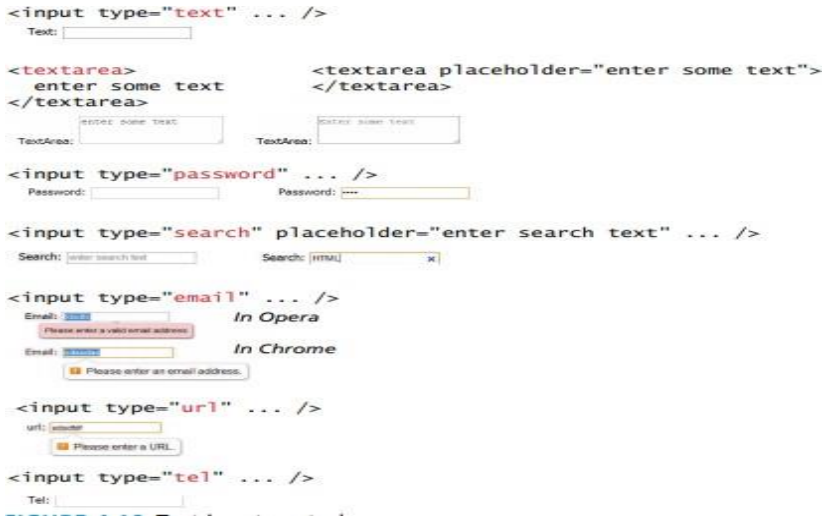


Figure 1.15: Text input controls

Figure 1.16: Using the pattern attribute

Choice Controls

- Forms often need the user to select an option from a group of choices. HTML provides several ways to do this.

Select Lists

- The <select> element is used to create a multiline box for selecting one or more items. The options can be hidden in a dropdown list or multiple rows of the list can be visible.
- Option items can be grouped together via the <optgroup> element. The selected attribute in the <option> makes it a default value. These options can be seen in Figure 1.17.

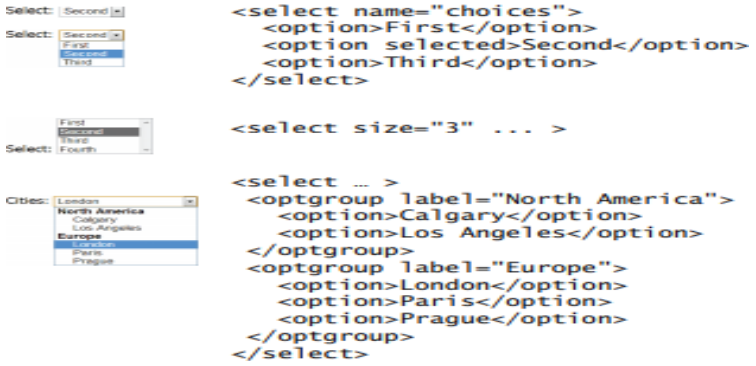


Figure 1.17: Using the <select> element

- The value attribute of the <option> element is used to specify what value will be sent back to the server in the query string when that option is selected.
- The value attribute is optional; if it is not specified, then the text within the container is sent instead, as can be seen in Figure 1.18.

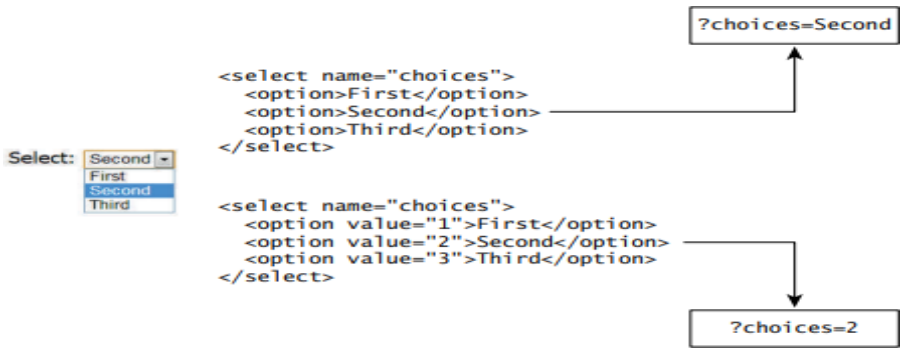


Figure 1.18: The value attribute

Radio Buttons

- **Radio buttons** are useful when we want the user to select a single item from a small list of choices and we want all the choices to be visible.
- As can be seen in Figure 1.19,radio buttons are added via the <input type="radio"> element.
- The buttons are made mutually exclusive (i.e., only one can be chosen) by sharing the same name attribute.

- The checked attribute is used to indicate the default choice, while the value attribute works in the same manner as with the <option> element.

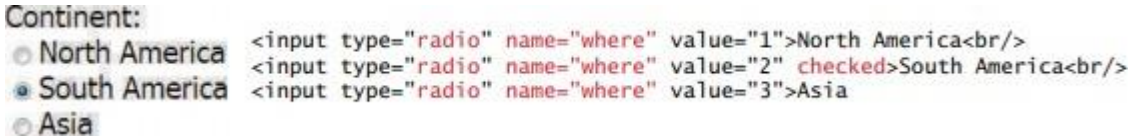


Figure 1.19: Radio buttons

Checkboxes

- **Checkboxes** are used for getting yes/no or on/off responses from the user. As can be seen in Figure 1.20, checkboxes are added via the <input type="checkbox">element.
- We can also group checkboxes together by having them share the same name attribute. Each checked checkbox will have its value sent to the server. Like with radio buttons, the checked attribute can be used to set the default value of a checkbox.

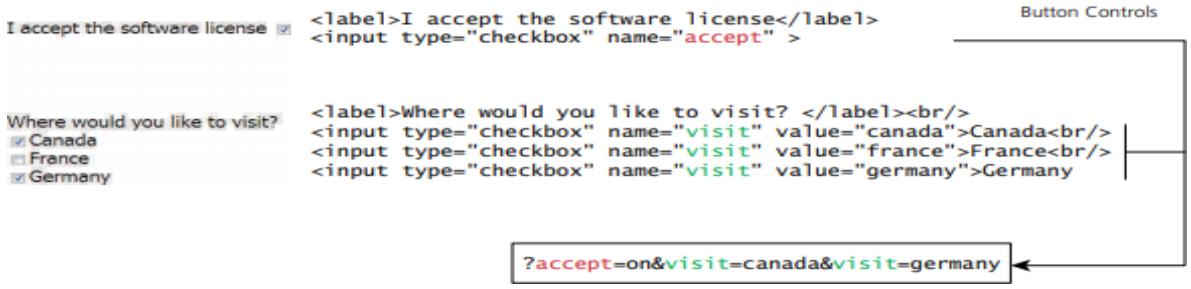


Figure 1.20: Checkbox buttons

Button Controls

- HTML defines several different types of buttons, which are shown in Table 1.4. As can be seen in that table, there is some overlap between two of the button types. Figure 1.21 demonstrates some sample button elements.

Type	Description
`<input type="submit">`	Creates a button that submits the form data to the server.
`<input type="reset">`	Creates a button that clears any of the user's already entered form data.
`<input type="button">`	Creates a custom button. This button may require JavaScript for it to actually perform any action.
`<input type="image">`	Creates a custom submit button that uses an image for its display.
`<button>`	Creates a custom button. The `<button>` element differs from `<input type="button">` in that you can completely customize what appears in the button; using it, you can, for instance, include both images and text, or skip server-side processing entirely by using hyperlinks. You can turn the button into a submit button by using the `type="submit"` attribute.

Table 1.4: Button Elements

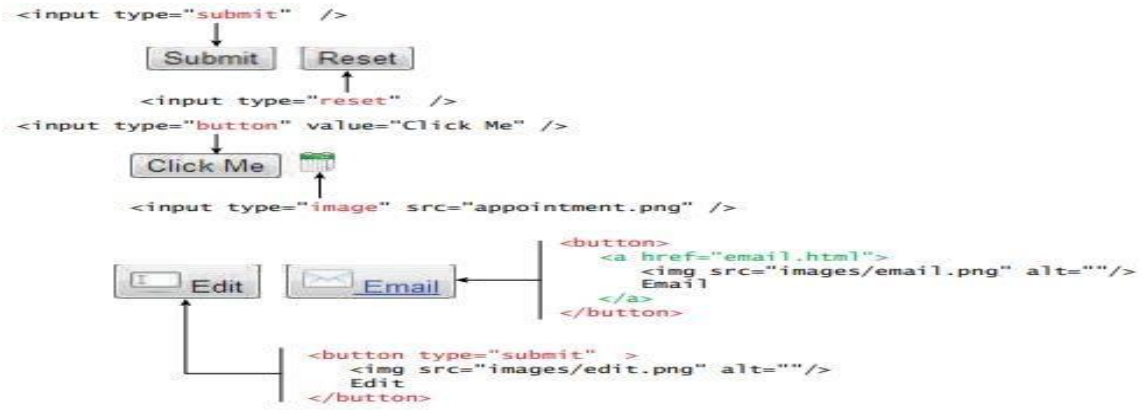


Figure 1.21: Example button elements

Specialized Controls

- There are two important additional special-purpose form controls that are available in all browsers.
- The specialized form control is the <input type="file"> element, which is used to upload a file from the client to the server.
- The usage and user interface for this control are shown in Figure 1.22.
- Notice that the <form> element must use the post method and that it must include the enctype="multipart/form-data" attribute as well.
- As we have seen in the section on query strings, form data is URL encoded (i.e., enctype="application/x-www-form-urlencoded"). However, files cannot be transferred to the server using normal URL encoding, hence the need for the alternative enctype attribute.

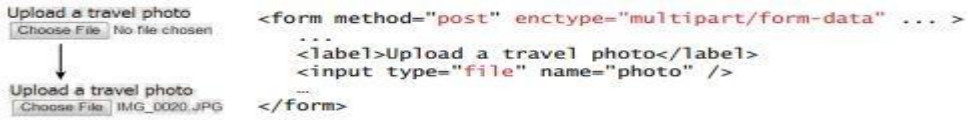


Figure 1.22: File upload control (in Chrome)

Number and Range

- When input via a standard text control, numbers typically require validation to ensure that the user has entered an actual number and, because the range of numbers is infinite, the entered number has to be checked to ensure it is not too small or too large.
- The number and range controls provide a way to input numeric values that eliminate the need for client-side numeric validation (for security reasons you would still check the

numbers for validity on the server). Figure 1.23 illustrates the usage and appearance of these numeric controls.

Figure 1.23: Number and range input controls

Color

- Not every web page needs the ability to get color data from the user, but when it is necessary, the HTML5 color control provides a convenient interface for the user, as shown in Figure 1.24.

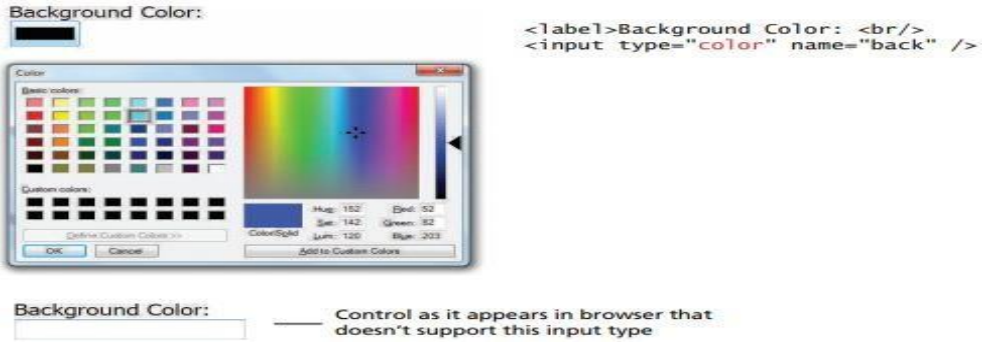


Figure 1.24: Color input control

Date and Time Controls

- Asking the user to enter a date or time is a relatively common web development task. Like with numbers, dates and times often need validation when gathering this information from a regular text input control.
- From a user's perspective, entering dates can be tricky as well: you probably have wondered at some point in time when entering a date into a web form, what format to enter it in, whether the day comes before the month, whether the month should be entered as an abbreviation or a number, and so on.
- Table 1.5 lists the various HTML5 date and time controls. Their usage and appearance in the browser are shown in Figure 1.25.

Type	Description
date	Creates a general date input control. The format for the date is "yyyy-mm-dd."
time	Creates a time input control. The format for the time is "HH:MM:SS," for hours:minutes:seconds.
datetime	Creates a control in which the user can enter a date and time.
datetime-local	Creates a control in which the user can enter a date and time without specifying a time zone.
month	Creates a control in which the user can enter a month in a year. The format is "yyyy-mm."
week	Creates a control in which the user can specify a week in a year. The format is "yyyy-W##."

Table 1.5: HTML5 Date and Time Controls

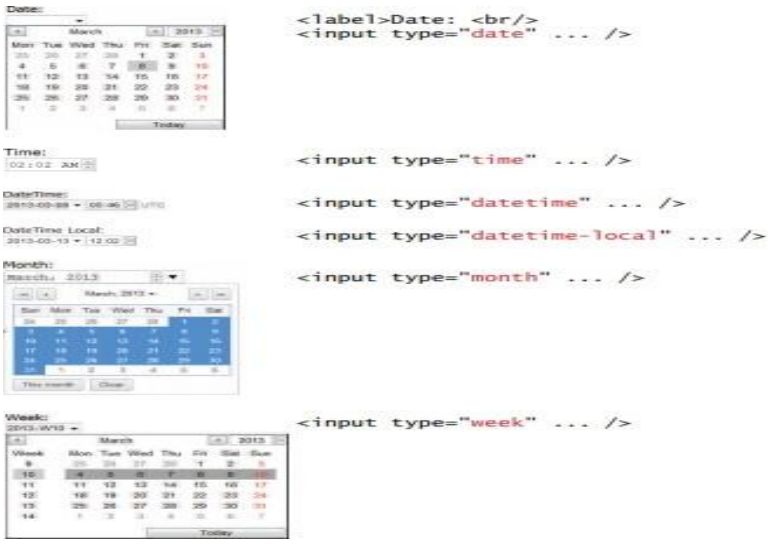


Figure 1.25: Date and time controls

Table and Form Accessibility

- **Problems:** Color blind users might have trouble differentiating certain colors in proximity; users with muscle control problems may have difficulty using a mouse, while older users may have trouble with small text and image sizes.
- The term **web accessibility** refers to the assistive technologies, various features of HTML that work with those technologies, and different coding and design practices that can make a site more usable for people with visual, mobility, auditory, and cognitive disabilities.
- In order to improve the accessibility of websites, the W3C created the **Web Accessibility Initiative (WAI)** in 1997. The WAI produces guidelines and recommendations, as well as organizing different working groups on different accessibility issues.
- One of its most helpful documents is the Web Content Accessibility. Guidelines, which is available at **http://www.w3.org/WAI/intro/wcag.php**.
- The most important guidelines in that document are:

- Provide text alternatives for any nontext content so that it can be changed into other forms people need, such as large print, braille, speech, symbols, or simpler language.
- Create content that can be presented in different ways (for example simpler layout) without losing information or structure.
- Make all functionality available from a keyboard.
- Provide ways to help users navigate, find content, and determine where they are.

Accessible Tables

- HTML tables can be quite frustrating from an accessibility standpoint. Users who rely on visual readers can find pages with many tables especially difficult to use. One vital way to improve the situation is to only use tables for tabular data, not for layout.
- Using the following accessibility features for tables in HTML can also improve the experience for those users:

1. Describe the table's content using the <caption> element.
    - This provides the user with the ability to discover what the table is about before having to listen to the content of each and every cell in the table.
2. Connect the cells with a textual description in the header. While it is easy for a sighted user to quickly see what row or column a given data cell is in, for users relying on visual readers, this is not an easy task.

```
<table>
    <caption>Famous Paintings</caption>
    <tr>
        <th scope="col">Title</th>
        <th scope="col">Artist</th>
        <th scope="col">Year</th>
        <th scope="col">Width</th>
        <th scope="col">Height</th>
    </tr>
    <tr>
        <td>The Death of Marat</td>
        <td>Jacques-Louis David</td>
        <td>1793</td>
        <td>162cm</td>
        <td>128cm</td>
    </tr>
    <tr>
        <td>Burial at ornans</td>
        <td>Gustave Courbet</td>
        <td>1849</td>
        <td>314cm</td>
        <td>663cm</td>
    </tr>
</table>
```

Listing 1.1: Connecting cells with headers

Accessible Forms

- HTML forms are also potentially problematic from an accessibility standpoint. If we remember the advice from the WAI about providing keyboard alternatives and text alternatives, our forms should be much less of a problem.
- The use of the <fieldset>, <legend>, and<label> elements, which provide a connection between the input elements in the form and their actual meaning.

```
<label for="f-title">Title: </label>

<input type="text" name="title" id="f-title"/>


<label for="f-country">Country: </label>

<select name="where" id="f-country">
    <option>Choose a country</option>
    <option>Canada</option>
    <option>Finland</option>
    <option>United States</option>
  </select>
```

Figure 1.26: Associating labels and input elements

- In other words, these controls add semantic content to the form and its logically group related form input elements together with the <legend> providing a type of caption for those elements.
- The <label> element has no special formatting. Each <label> element should be associated with a single input element. i.e., if the user clicks on or taps the <label> text, that control will receive the form's focus.

Microformats

- Most sites have some type of Contact Us page, in which addresses and other information are displayed; similarly, many sites contain a calendar of upcoming events or information about products or news.
- The idea behind Microformats is that if this type of common information were tagged in similar ways, then automated tools would be able to gather and transform it.
- Thus, a **microformat** is a small pattern of HTML markup and attributes to represent common blocks of information such as people, events, and news stories so that the information in them can be extracted and indexed by software agents.
- Figure 1.27 illustrates this process. One of the most common Microformats is **hCard**, which is used to semantically markup contact information for a person. Google Map search results now make use of the hCard microformat.

- Listing 1.2 illustrates the example markup for a person's contact information that uses the hCard microformat. To learn more about the hCard format, visit **http://microformats.org/wiki/hcard**.

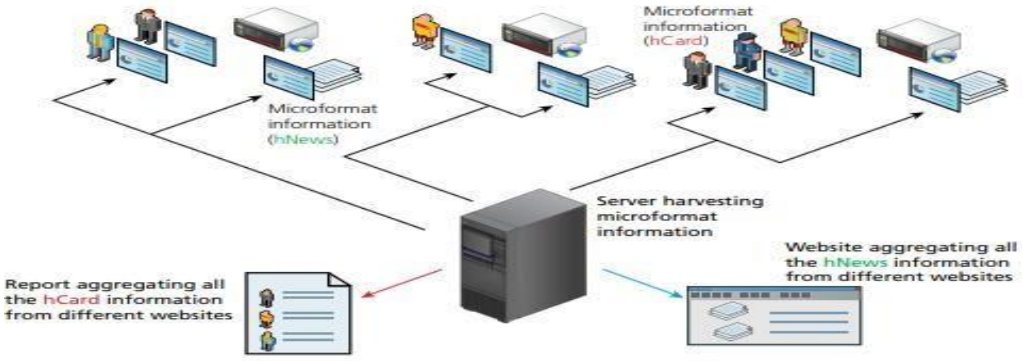


Figure 1.27: Microformats

```
<div class="vcard">
    <span class="fn">Randy Connolly</span>
    <div class="org">Mount Royal University</div>
    <div class="adr">
        <div class="street-address">4825 Mount Royal Gate SW</div>
        <div>
           <span class="locality">Calgary</span>,
           <abbr class="region" title="Alberta">AB</abbr>
           <span class="postal-code">T3E 6K6</span>
        </div>
        <div class="country-name">Canada</div>
    </div>
    <div>Phone: <span class="tel">+1-403-440-6111</span></div>
</div>
```

Listing 1.2: Example of an hCard

Chapter 2: Advanced CSS-Layout

1. Normal Flow
2. Positioning Elements
3. Floating Elements
4. Constructing Multicolumn Layouts
5. Approaches to CSS Layout
6. Responsive Design
7. CSS Frameworks

Normal Flow

- **Normal flow**, which refers here to how the browser will normally display block-level elements and inline elements from left to right and from top to bottom.

Block-level elements

- <p>, , <h2>, <ul>, and <table> are each contained on their own line. Because block-level elements begin with a line break (that is, they start on a new line).
- Without styling, two block-level elements can't exist on the same line.
- Block-level elements use the normal CSS boxmodel, in that they have margins, paddings, background colors, and borders.

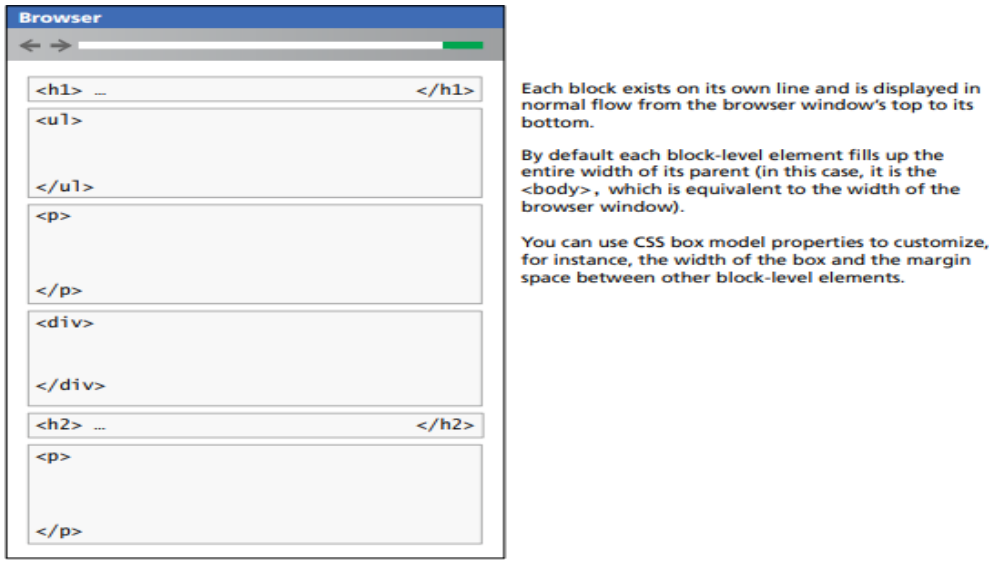


Figure 2.1: Block-level elements

Inline elements

- These elements do not form their own blocks but instead are displayed within lines.
- Normal text in an HTML document is inline, as are elements such as ,<a>, <img>, and .
- Inline elements line up next to one another horizontally from left to right on the same line.
- When there isn't enough space left on the line, the content moves to a new line, as shown in Figure 2.2.

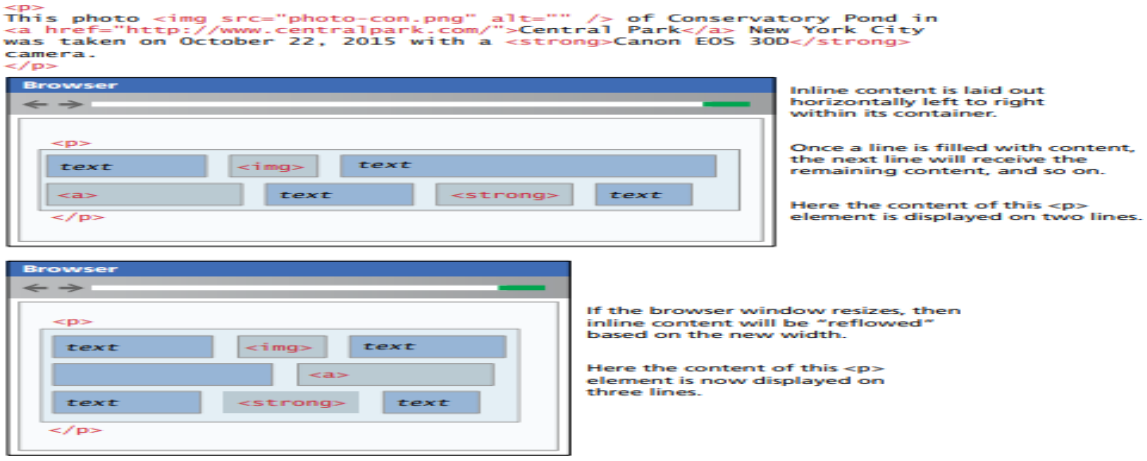


Figure 2.2: Inline elements

- There are actually two types of inline elements: replaced and nonreplaced.

Replaced inline elements

- These are elements whose content is defined by some external resource, such as <img> and the various form elements.
- Replaced inline elements have a width and height that are defined by the external resource

Nonreplaced inline elements

- These are elements whose content is defined within the document, which includes all the other inline elements.
- Width of these elements are defined by their content (and by other properties such as font-size and letter-spacing), the width property is ignored, as are the margin-top, margin-bottom, and the height.

- In a document with normal flow, block-level elements and inline elements work together as shown in Figure 2.3. Block-level elements will flow from top to bottom, while inline elements flow from left to right within a block.
- It is possible to change whether an element is block-level or inline via the CSS display property. Consider the following two CSS rules:

```
span { display: block; }
li { display: inline; }
```

- These two rules will make all elements behave like block-level elements and all <li> elements like inline (that is, each list item will be displayed on the same line).

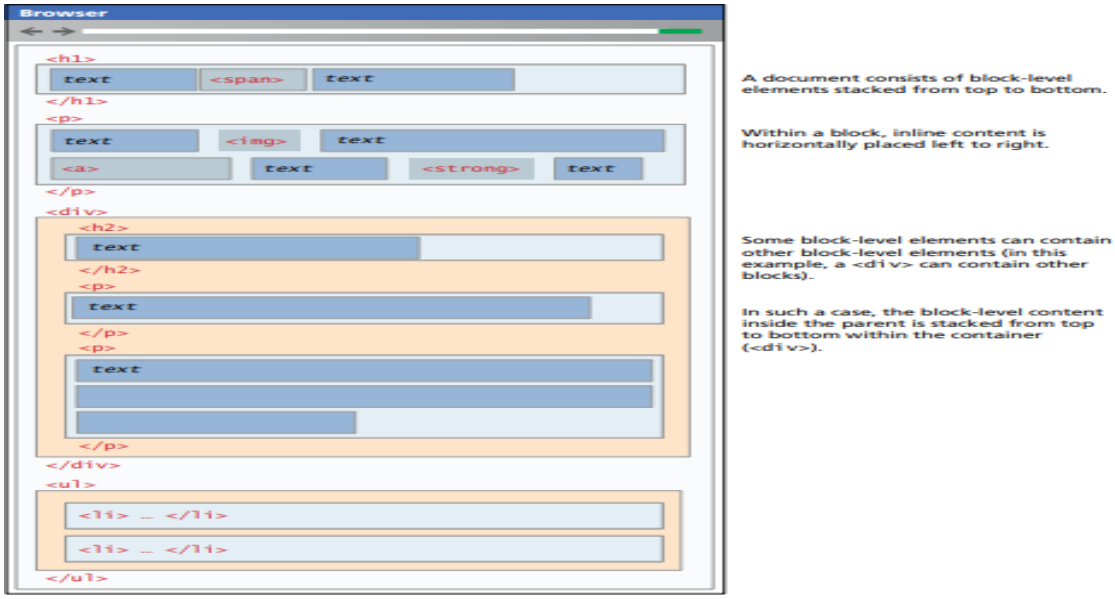


Figure 2.3: Block and inline elements together

Positioning Elements

- It is possible to move an item from its regular position in the normal flow, and even move an item outside of the browser viewport so it is not visible or to position it so, it is always visible in a fixed position while the rest of the content scrolls.
- The position property is used to specify the type of positioning, and the possible values are shown in Table 2.1.
- The left, right, top, and bottom properties are used to indicate the distance the element will move; the effect of these properties varies depending upon the position property.

Value	Description
absolute	The element is removed from normal flow and positioned in relation to its nearest positioned ancestor.
fixed	The element is fixed in a specific position in the window even when the document is scrolled.
relative	The element is moved relative to where it would be in the normal flow.
static	The element is positioned according to the normal flow. **This is the default.**

Table 2.1: Position Values

Relative Positioning

- In **relative positioning** an element is displaced out of its normal flow position and moved relative to where it would have been placed.
- The other content around the relatively positioned element "remembers" the element's old position in the flow; thus the space the element would have occupied is preserved as shown in Figure 2.4.
- The positioned element now overlaps other content: that is, the <p> element following the <figure> element does not change to accommodate the moved<figure>.

```
<p>A wonderful serenity has taken possession of my …
<figure>
    <img src="images/828.jpg" alt="" />
    <figcaption>British Museum</figcaption>
</figure>
<p>When, while the lovely valley …
```


```
figure {
    border: 1pt solid #A8A8A8;
    background-color: #EDEDDD;
    padding: 5px;
    width: 150px;
    position: relative;
    top: 150px;
    left: 200px;
}
```

Figure 2.4: Relative positioning

Absolute Positioning

- When an element is positioned absolutely, it is removed completely from normal flow. Thus, unlike with relative positioning, space is not left for the moved element, as it is no longer in the normal flow.
- Its position is moved in relation to its container block. In the example shown in Figure 2.5, the container block is the <body> element. Like with the relative positioning example, the moved block can now overlap content in the underlying normal flow.

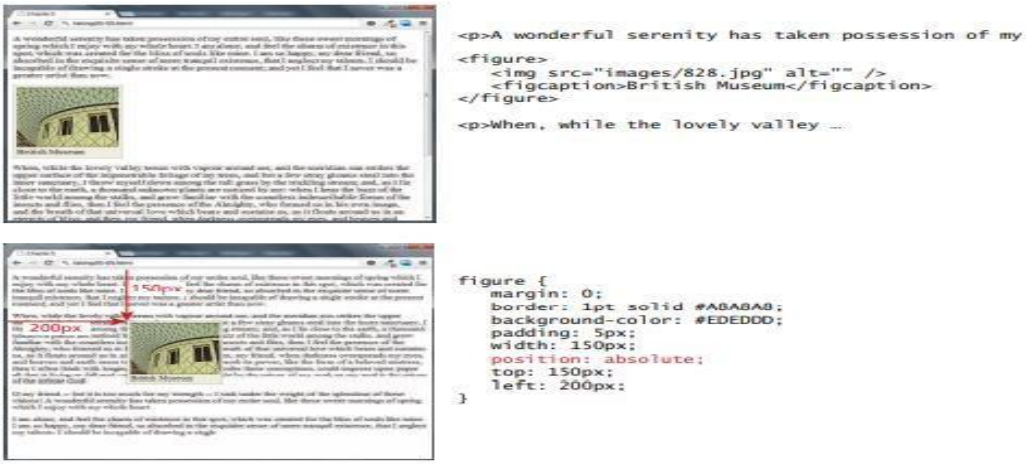


Figure 2.5: Absolute positioning

- A moved element via absolute position is actually positioned relative to its nearest **positioned** ancestor container (that is, a block-level element whose position is fixed, relative, or absolute).
- In the example shown in Figure 2.6, the <figcaption> is absolutely positioned; which happens to be its parent (the <figure> element).

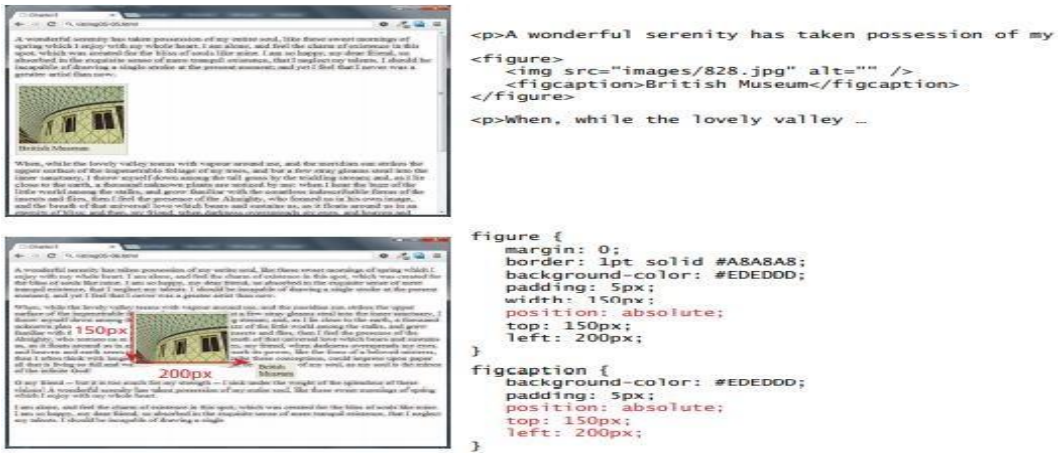


Figure 2.6: Absolute position is relative to nearest positioned ancestor container.

Z-Index

- Each positioned element has a stacking order defined by the z-index property (named for the z-axis). Items closest to the viewer (and thus on the top) have a larger **z-index** value.

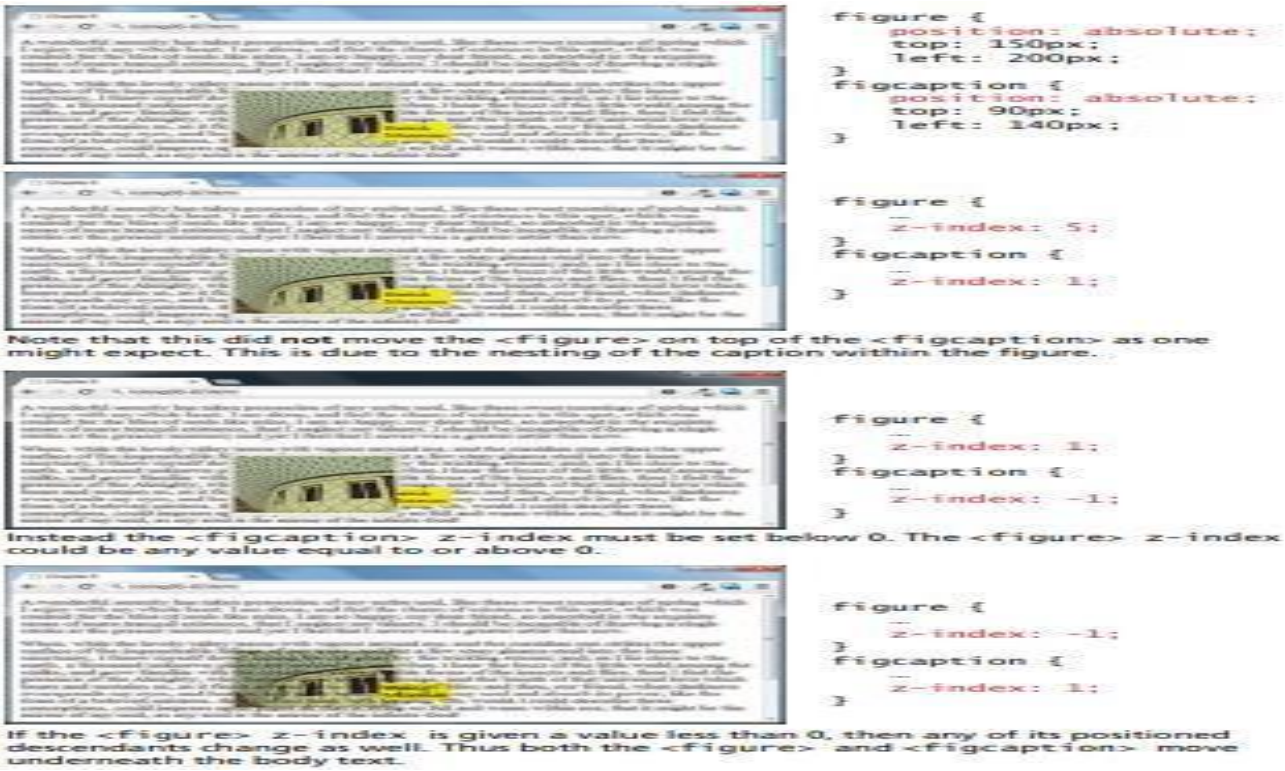


Figure 2.7: Z-index

Fixed Position

- The fixed position value is used relatively infrequently. It is a type of absolute positioning, except that the positioning values are in relation to the viewport (i.e., to the browser window).
- Elements with **fixed positioning** do not move when the user scrolls up or down the page.
- The fixed position is most commonly used to ensure that navigation elements or advertisements are always visible.

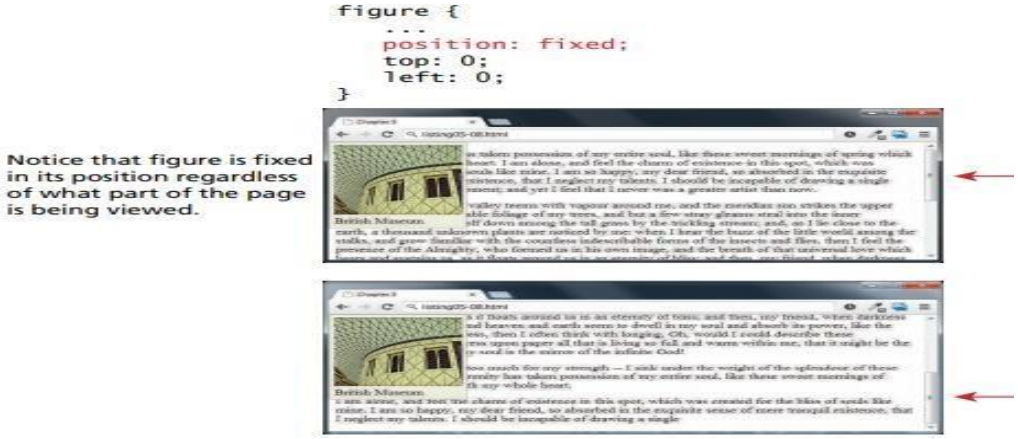


Figure 2.8: Fixed position

Floating Elements

- It is possible to displace an element out of its position in the normal flow via the CSS float **property**.
- An element can be floated to the left or floated to the right. When an item is floated, it is moved all the way to the far left or far right of its containing block and the rest of the content is "re-flowed" around the floated element, as can be seen in Figure 2.9.

Figure 2.9: Floating an element

- The floated block-level element must have a width specified; if we don't, then the width will be set to auto, which will mean it implicitly fills the entire width of the containing block, and there thus will be no room available to flow content around the floated item.

Floating within a Container

- It should be reiterated that a floated item moves to the left or right of its container (also called its **containing block**).
- In Figure 2.9, the containing block is the HTML document itself so the figure moves to the left or right of the browser window.
- Figure 2.10, the floated figure is contained within an <article> element that is indented from the browser's edge. The relevant margins and padding areas are color coded to help make it clearer how the float interacts with its container.
- In this illustration, we can see that the overlapping margins for the adjacent <p> elements behave normally and collapse. But the top margin for the floated<figure> and the bottom margin for the <p> element above it do *not* collapse.

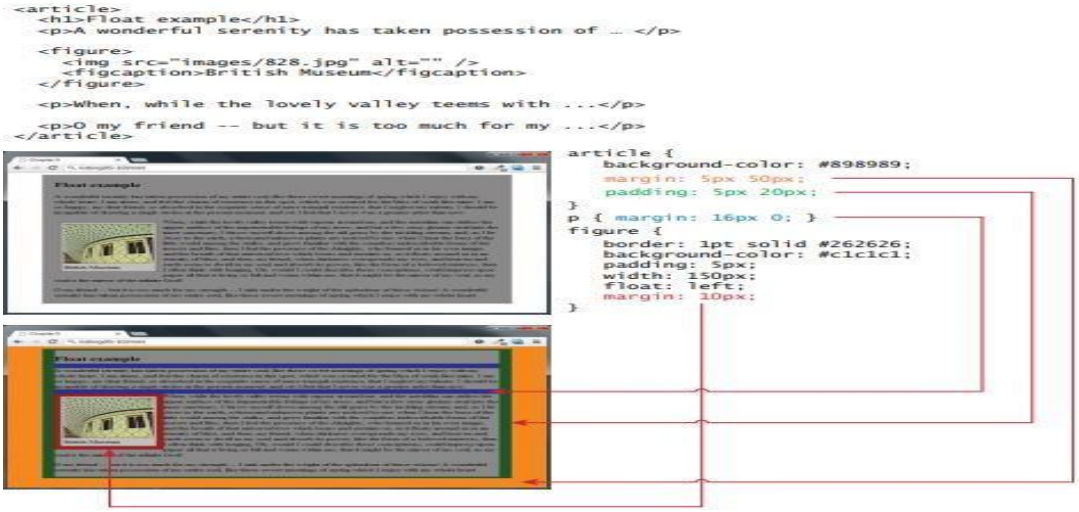


Figure 2.10: Floating to the containing block

Floating Multiple Items Side by Side

- One of the more common usages of floats is to place multiple items side by side on the same line.
- When we float multiple items that are in proximity, each floated item in the container will be nestled up beside the previously floated item.
- All other content in the containing block (including other floated elements) will flow around all the floated elements, as shown in Figure 2.11.

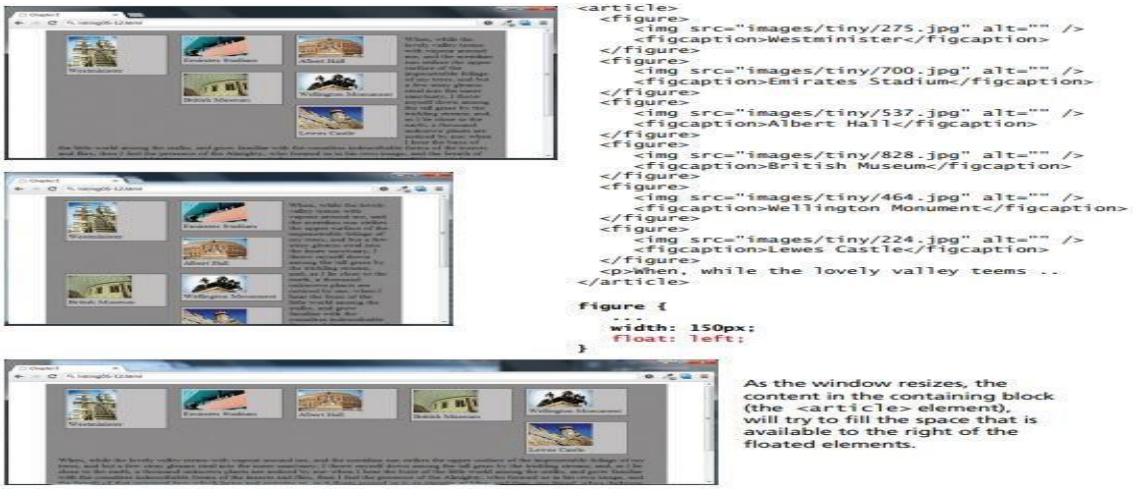


Figure 2.11: Problems with multiple floats

Problems

- As the browser window increases or decreases in size (that is, as the containing block resizes).

Solution:

- We can stop elements from flowing around a floated element by using the clear property.
- The values for this property are shown in Table 2.2. Figure 2.12 demonstrates how the use of the clear property.

Value	Description
`left`	The left-hand edge of the element cannot be adjacent to another element.
`right`	The right-hand edge of the element cannot be adjacent to another element.
`both`	Both the left-hand and right-hand edges of the element cannot be adjacent to another element.
`none`	The element can be adjacent to other elements.

Table 2.2: clear property

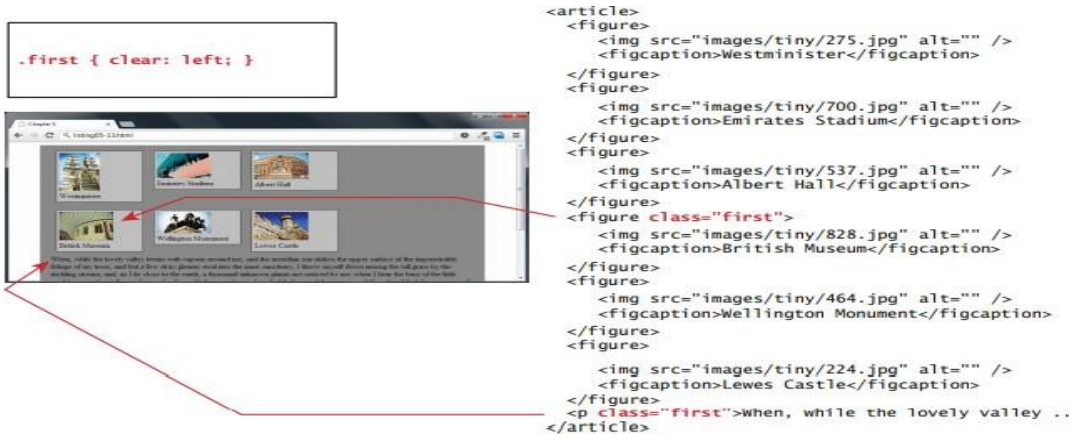


Figure 2.12: Using the clear property

Containing Floats

- Another problem that can occur with floats is when an element is floated within a containing block that contains only floated content. In such a case, the containing block essentially disappears, as shown in Figure 2.13.

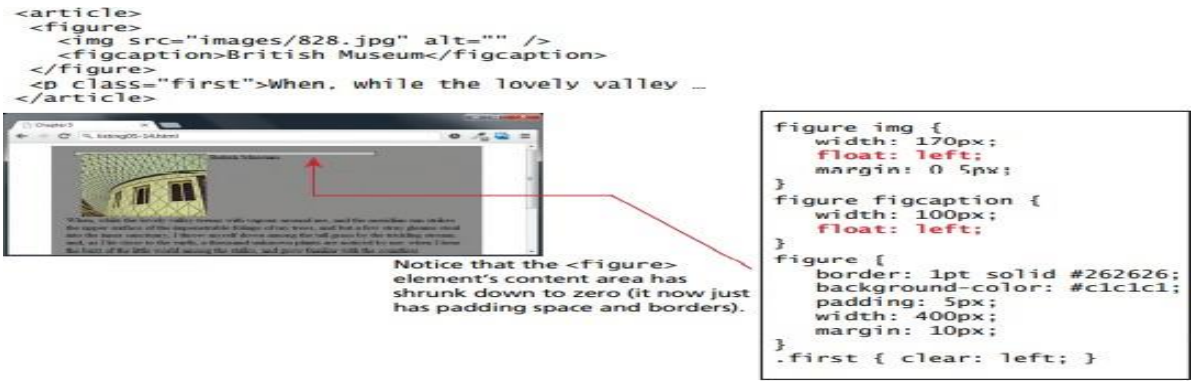


Figure 2.13: Disappearing parent containers

- In Figure 2.13, the <figure> containing block contains only an <img> and <figcaption> element, and both of these elements are floated to the left. i.e., both elements have been removed from the normal flow; from the browser's perspective, since the <figure> contains no normal flow content, it essentially has nothing in it, hence it has a content height of zero. To avoid this a better solution would be to use the **overflow property** as shown in Figure 2.14.

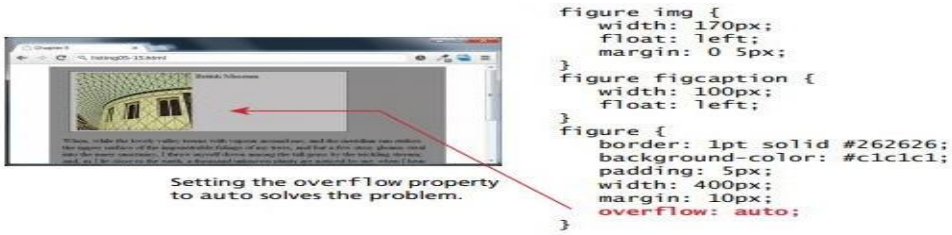


Figure 2.14: Using the overflow property

Overlaying and Hiding Elements

- One of the more common design tasks with CSS is to place two elements on top of each other, or to selectively hide and display elements.
- Positioning is often used for smaller design changes, such as moving items relative to other elements within a container. In such a case, relative positioning is used to create the **positioning context** for a subsequent absolute positioning move.

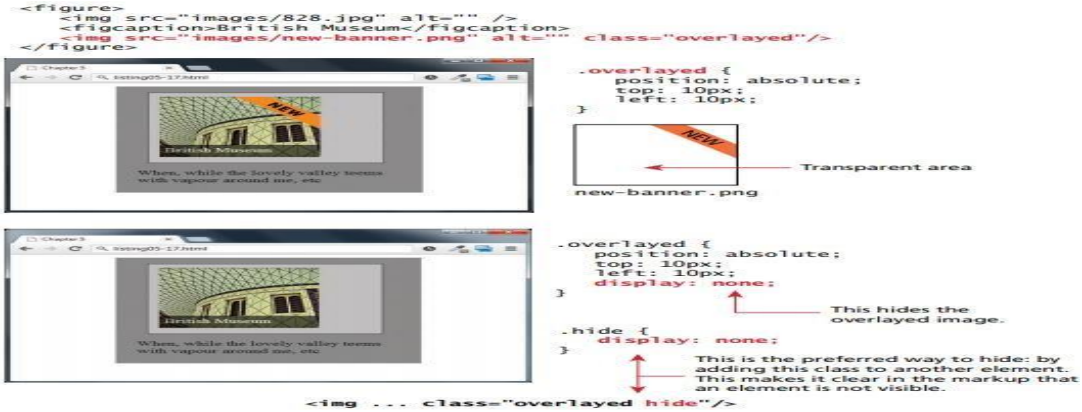


Figure 2.15: Using the display property

- An image that is the same size as the underlying one is placed on top of the other image using absolute positioning. Since most of this new image contains transparent pixels, it only covers part of the underlying image.

Figure 2.16: Comparing display to visibility

- There are in fact two different ways to hide elements in CSS: using the display property and using the visibility property.
  - The **display property** takes an item out of the flow: it is as if the element no longer exists.
  - The **visibility property** just hides the element, but the space for that element remains.
- While these two properties are often set programmatically via JavaScript, it is also possible to make use of these properties without programming using the :hover pseudo-class.
- Figure 2.17 demonstrates how the combination of absolute positioning, the :hover pseudo-class, and the visibility property can be used to display a larger version of an image when the mouse hovers over the thumbnail version of the image.

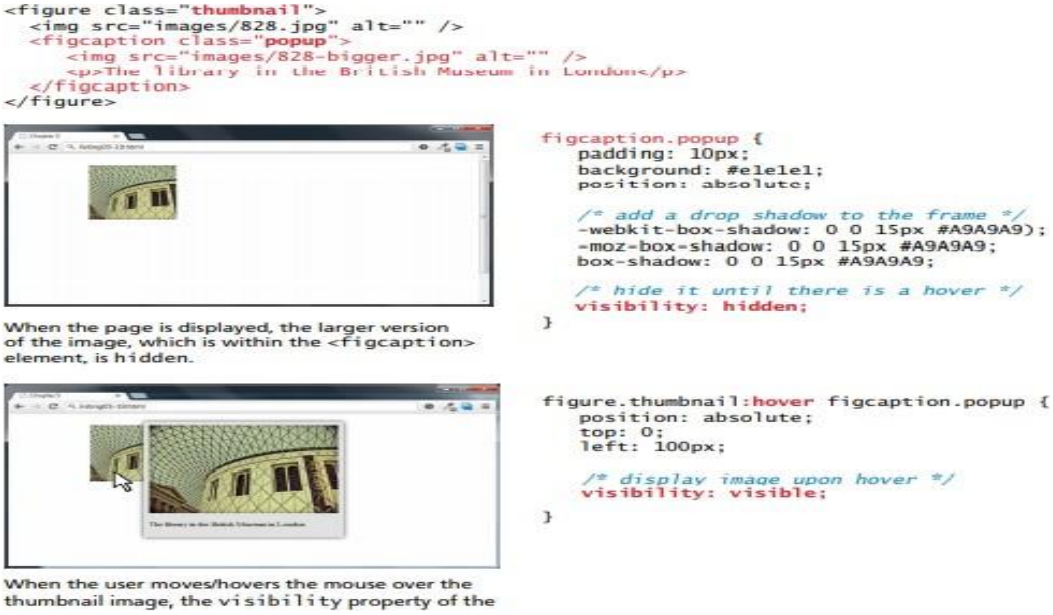


Figure 2.17: Using hover with display

Constructing Multicolumn Layouts

They are the raw techniques that we can use to create more complex layouts. A typical layout may very well use both positioning and floats.

Using Floats to Create Columns

- Using floats is perhaps the most common way to create columns of content.
- The first step is to float the content container that will be on the left-hand side. Remember that the floated container needs to have a width specified.
- As can be seen in the second screen capture in Figure 2.18, the other content will flow around the floated element.
- Figure 2.18 shows the other key step: changing the left-margin so that it no longer flows back under the floated content.

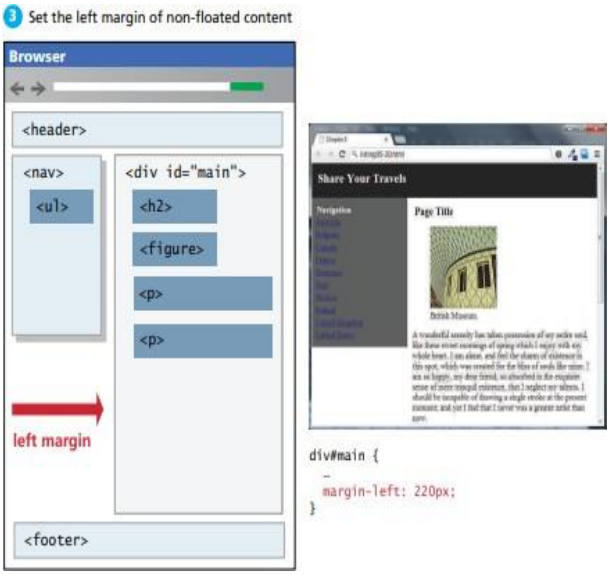


Figure 2.18: Creating two-column layout

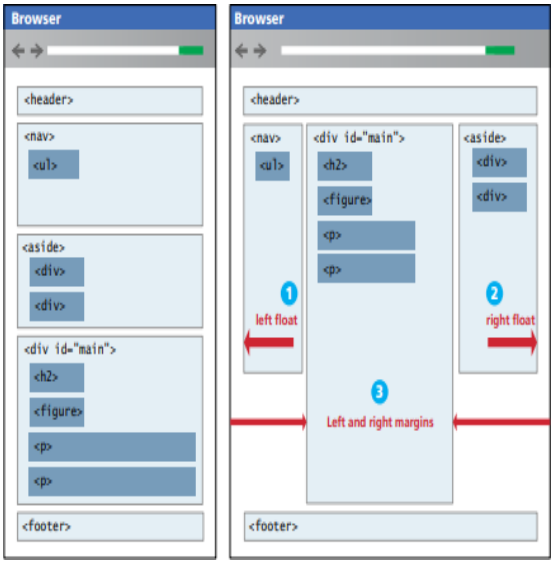


Figure 2.19: Creating a three-column layout

Issues:

- The background of the floated element stops when its content ends. If we wanted the background color to descend down to the footer, then it is difficult (but not impossible) to achieve this visual effect with floats.

- A three-column layout could be created in much the same manner, as shown in Figure 2.19 page layouts.
- Another approach for creating a three-column layout is to float elements *within* a container element.

- This approach is actually a little less brittle because the floated elements within a container are independent of elements outside the container.

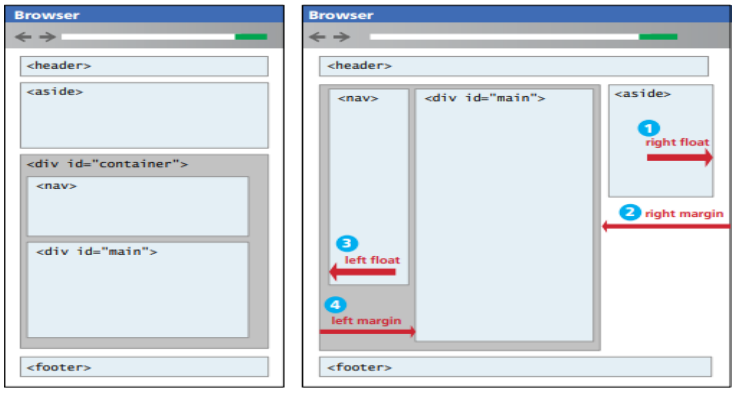


Figure 2.20: Creating a three-column layout with nested floats

- Notice again that the floated content must appear in the source *before* the non-floated content. This is the main problem with the floated approach.

Using Positioning to Create Columns

- Positioning can also be used to create a multicolumn layout. Typically, the approach will make use of absolute position.
- Figure 2.21 illustrates a typical three-column layout implemented via positioning. Notice that with positioning it is easier to construct our source document with content in a more SEO-friendly manner; in this case, the main can be placed first.

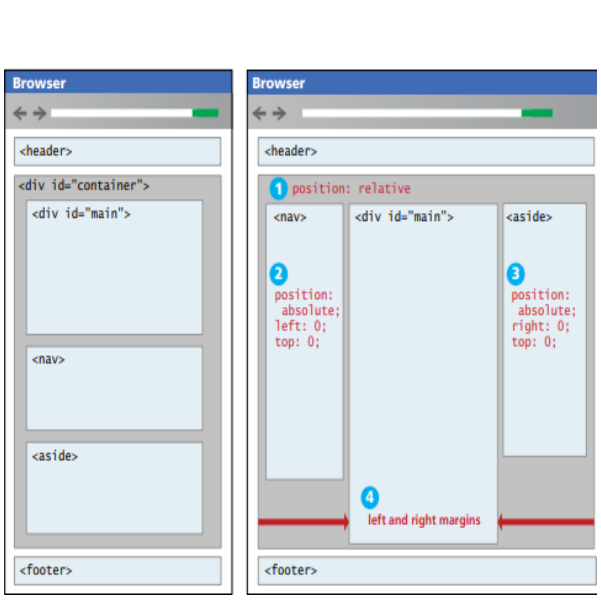


Figure 2.21: Three-column layout with positioning

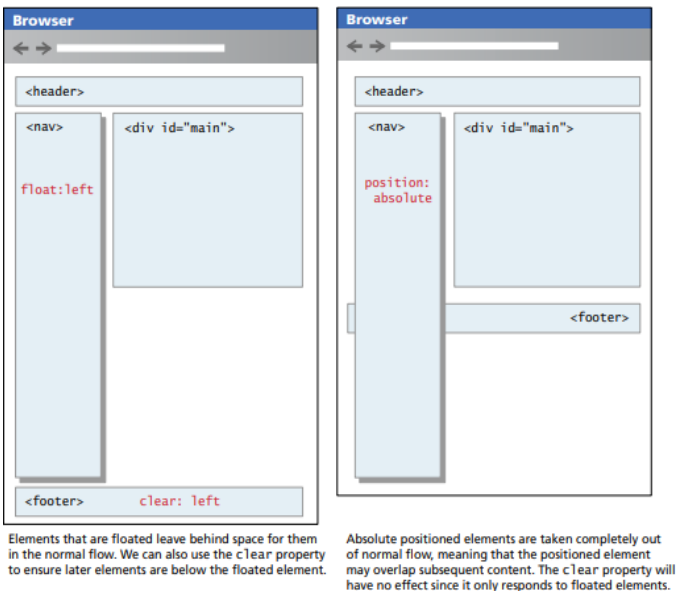


Figure 2.22: Problems with absolute positioning

Problems

- What would happen if one of the sidebars had a lot of content and was thus quite long?
  - In the floated layout, this would not be a problem at all, because when an item is floated, blank space is left behind.
  - But when an item is positioned, it is removed entirely from normal flow, so subsequent items will have no "knowledge" of the positioned item. This problem is illustrated in Figure 2.22.
- One solution to this type of problem is to place the footer within the main container.

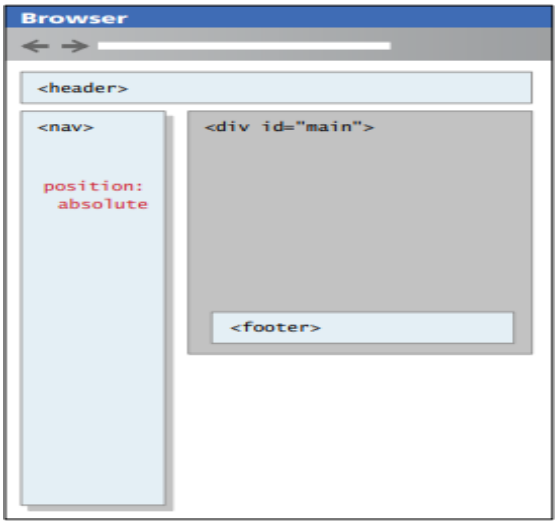


Figure 2.23: Solution to the footer problem

Approaches to CSS Layout

- One of the main problems faced by web designers is that the size of the screen used to view the page can vary quite a bit.
  - Some users will visit a site on a 21-inch wide screen monitor that can display 1920 × 1080 pixels (px);
  - Others will visit it on an older iPhone with a 3.5 screen and a resolution of 320 × 480 px.
  - Users with the large monitor might expect a site to take advantage of the extra size;
  - Users with the small monitor will expect the site to scale to the smaller size and still be usable.

- Satisfying both users can be difficult; the approach to take for one type of site content might not work as well with another site with different content.

- The two basic approaches to deal with the problems of screen size are **Fixed** and **Liquid** layouts.

Fixed Layout

- In a **fixed layout**, the basic width of the design is set by the designer, typically corresponding to an "ideal" width based on a "typical" monitor resolution.
- A common width used is something in the 960 to1000 pixel range, which fits nicely in the common desktop monitor resolution (1024 × 768). This content can then be positioned on the left or the center of the monitor.
- Fixed layouts are created using pixel units, typically with the entire content within a container whose width property has been set to some width as shown in figure 2.24.

  Advantage

  - It is easier to produce and generally has a predictable visual result.
  - It is also optimized for typical desktop monitors; however, as more and more user visits are happening via smaller mobile devices, this advantage might now seem to some as a disadvantage.

  Disadvantage

  - Shrinks below the fixed width; the user will have to horizontally scroll to see all the content.

- Fixed layouts have other **drawbacks.** For larger screens, there may be an excessive amount of blank space to the left and/or right of the content. Much worse is when the browser window shrinks below the fixed width; the user will have to horizontally scroll to see all the content, as shown in Figure 2.25.

Figure 2.24: Fixed layouts

Figure 2.25: Problems with fixed layout

Liquid Layout

- The second approach to dealing with the problem of multiple screen sizes is to use a **liquid layout** (also called a **fluid layout**). In this approach, widths are not specified using pixels, but percentage values.
- CSS are a percentage of the current browser width, so a layout in which all widths are expressed as percentages should adapt to any browser size, as shown in Figure 2.26.

Advantage

- It adapts to different browser sizes, so there is neither wasted white space nor any need for horizontal scrolling.

Disadvantages

- Liquid layouts can be more difficult to create because some elements, such as images, have fixed pixel sizes.
- As the screen grows or shrinks dramatically, in that the line length may become too long or too short.

Figure 2.26: Liquid Layouts

Other Layout Approaches

- While the fixed and liquid layouts are the two basic paradigms for page layout, there are some other approaches that combine the two layout styles i.e., **Hybrid layout (Combination pixels and percentages).**
- Fixed pixel measurements might make sense for a sidebar column containing mainly graphic advertising images that must always be displayed and which always are the same width.
- But percentages would make more sense for the main content or navigation areas, with perhaps min and max size limits in pixels set for the navigation areas.

Responsive Design

- **Problems of a liquid layout** is that images and horizontal navigation elements tend to take up a fixed size, and when the browser window shrinks to the size of a mobile browser, liquid layouts can become unusable.
- In a **responsive design**, the page "responds" to changes in the browser size that go beyond the width scaling of a liquid layout.

- In a responsive layout, images will be scaled down and navigation elements will be replaced as the browser shrinks, as can be seen in Figure 2.27.

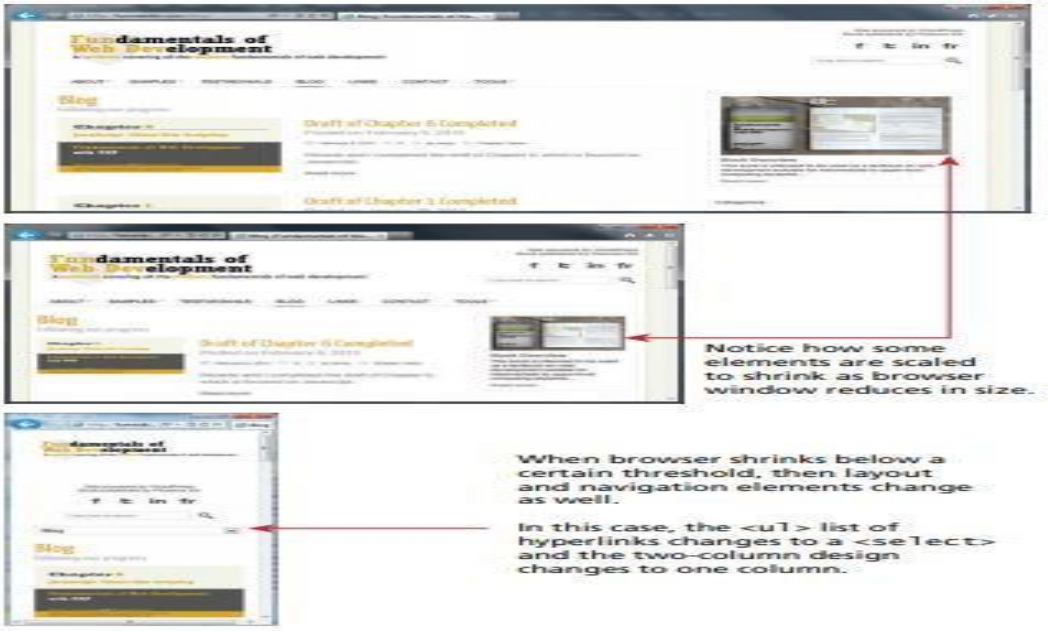


Figure 2.27: Responsive layouts

- There are four key components that make responsive design work. They are:
  1. Liquid layouts
  2. Scaling images to the viewport size
  3. Setting viewports via the <meta> tag
  4. Customizing the CSS for different viewports using media queries.
- Responsive designs begin with a liquid layout, that is, one in which most elements have their widths specified as percentages.
- Making images scale in size is actually quite straightforward, in that you simply need to specify the following rule:

```
img {
        max-width: 100%;
}
```

- Of course this does not change the downloaded size of the image; it only shrinks or expands its visual display to fit the size of the browser window, never expanding beyond its actual dimensions.

Setting Viewports

- A key technique in creating responsive layouts makes use of the ability of current mobile browsers to shrink or grow the web page to fit the width of the screen. If we have ever used a modern mobile browser, we may have been surprised to see how the web page was scaled to fit into the small screen of the browser.
- The way this works is the mobile browser renders the page on a canvas called the **viewport**.

  Ex: On iPhones, for instance, the viewport width is 980 px, and then that viewport is scaled to fit the current width of the device as shown in Figure 2.28.

Figure 2.28: viewports

- The mobile Safari browser introduced the viewport <meta> tag as a way for developers to control the size of that initial viewport.
- The web page can tell the mobile browser the viewport size to use via the viewport <meta> element, as shown in Listing 2.1.

```
<html>
<head>
<meta name="viewport" content="width=device-width" />
```

Listing 2.1: Setting the viewports

- By setting the viewport as in this listing, the page is telling the browser that no scaling is needed, and to make the viewport as many pixels wide as the device screen width.
- This means that if the device has a screen that is 320 px wide, the viewport width will be 320 px; if the screen is 480 px then the viewport width will be 480 px. The result will be similar to that shown in Figure 2.29

Figure 2.29: setting the viewports

- There needs to be a way to transform the look of the site for the smaller screen of the mobile device, which is the job of the next key component of responsive design, media queries.

Media Queries

- The other key component of responsive designs is **CSS media queries**.
- A media query is a way to apply style rules based on the medium that is displaying the file.
- We can use these queries to look at the capabilities of the device, and then define CSS rules to target that device. Unfortunately, media queries are not supported by Internet Explorer 8 and earlier.

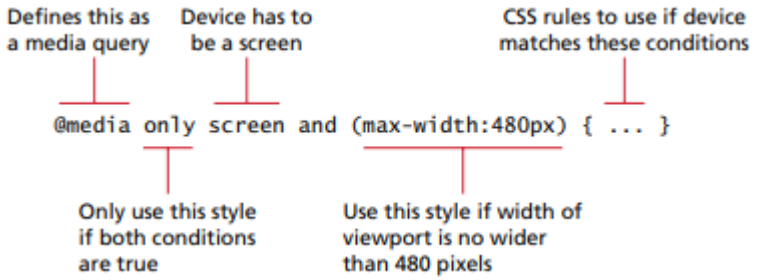


Figure 2.30: Sample media query

- Figure 2.30 illustrates the syntax of a typical media query. These queries are Boolean expressions and can be added to our CSS files or to the <link> element to conditionally use a different external CSS file based on the capabilities of the device.
- Table 2.3 is a partial list of the browser features you can examine with media queries. Many of these features have min- and max- versions.

Feature	Description
width	Width of the viewport
height	Height of the viewport
device-width	Width of the device
device-height	Height of the device
orientation	Whether the device is portrait or landscape
color	The number of bits per color

Table 2.3: Browser Features You Can Examine with Media Queries

- Contemporary responsive sites will typically provide CSS rules for phone displays first, then tablets, then desktop monitors, an approach called **progressive enhancement**

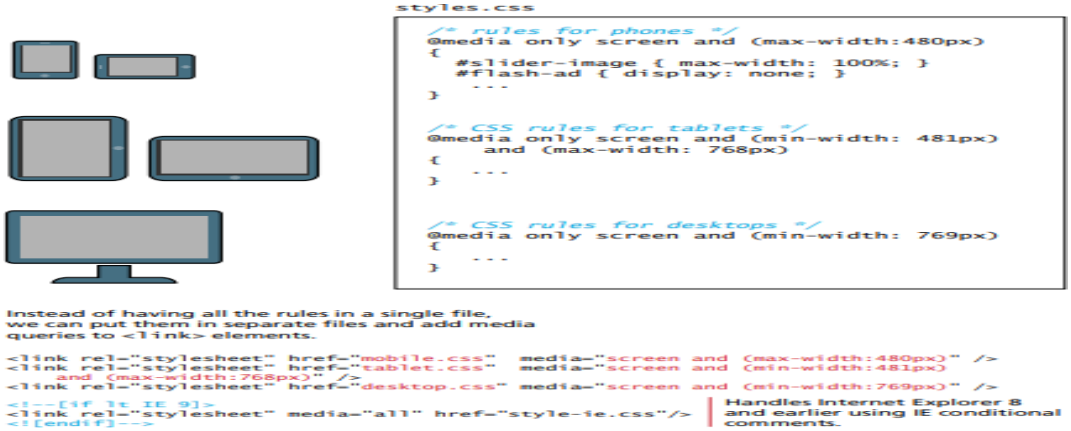


Figure 2.31: Media queries in action

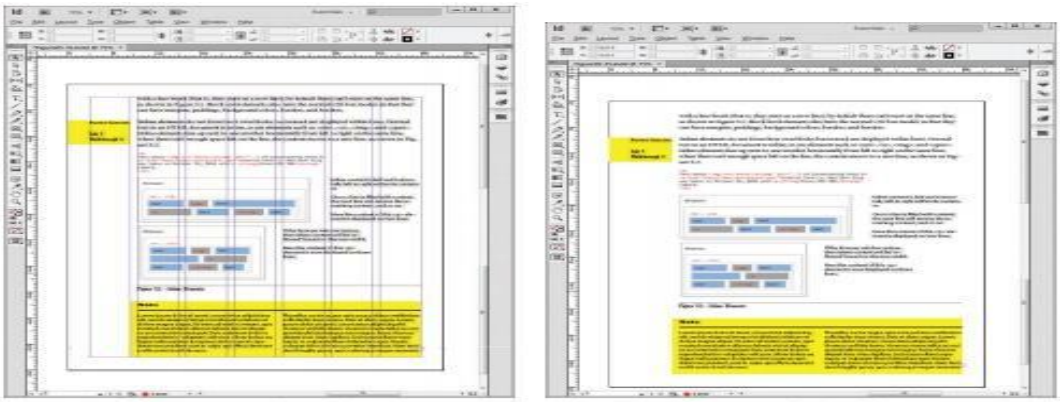
- Since later rules override earlier rules, this. provides progressive enhancement, meaning that as the device grows we can have CSS rules that take advantage of the larger space. Notice as well that these media queries can be within our CSS file or within the <link> element.

CSS Frameworks

- A **CSS framework** is a pre created set of CSS classes or other software tools that make it easier to use and work with CSS.
- They are two main types of CSS framework: grid systems and CSS preprocessors.

Grid Systems

- **Grid systems** make it easier to create multicolumn layouts.
- There are many CSS grid systems; some of the most popular are Bootstrap (**twitter.github.com/bootstrap**),Blueprint (**www.blueprintcss.org**), and 960 (**960.gs**).
- Print designers typically use grids as a way to achieve visual uniformity in a design. In print design, the very first thing a designer may do is to construct, for instance, a 5- or 7- or 12-column grid in a page layout program like InDesign or Quark Xpress. The rest of the document, whether it be text or graphics, will be aligned and sized according to the grid, as shown in Figure 2.32.



Most page design begins with a grid. In this case, a seven-column grid is being used to layout page elements in Adobe InDesign.

Without the gridlines visible, the elements on the page do not look random, but planned and harmonious.

Figure 2.32: Using a grid in print design

- CSS frameworks provide similar grid features.
  - The 960 framework uses either a 12- or 16-column grid.
  - Bootstrap uses a 12-column grid. Blueprint uses a 24-column grid.
- The grid is constructed using elements with classes defined by the framework. The HTML elements for the rest of your site are then placed within these elements.
- Listing 2.2 illustrates a three column layout within the grid system of the 960 framework.
- Listing 2.3 shows the same thing in the Bootstrap framework.
- In both systems, elements are laid out in rows; elements in a row will span from 1to 12 columns.
- In the 960 system, a row is terminated with .
- In Bootstrap, content must be placed within the row container.

```
<head>
  <link rel="stylesheet" href="reset.css" />
  <link rel="stylesheet" href="text.css" />
  <link rel="stylesheet" href="960.css" />
</head>
<body>
  <div class="container_12">
    <div class="grid_2">
      left column
    </div>
    <div class="grid_7">
      main content
    </div>
    <div class="grid_3">
      right column
    </div>
    <div class="clear"></div>
  </div>
</body>
```

Listing 2.2: Using the 960 grid

```
<head>
   <link href="bootstrap.css" rel="stylesheet">
</head>
<body>
  <div class="container">
    <div class="row">
      <div class="col-md-2">
        left column
      </div>
      <div class="col-md-7">
         main content
      </div>
      <div class="col-md-3">
        right column
      </div>
    </div>
  </div>
</body>
```

Listing 2.3: Using the 960 grid

- Both of these frameworks allow columns to be nested, making it quite easy to construct the most complex of layouts.
- CSS frameworks may reduce your ability to closely control the styling on your page, and conflicts may occur when multiple CSS frameworks are used together.

CSS Preprocessors

- **CSS preprocessors** are tools that allow the developer to write CSS that takes advantage of programming ideas such as variables, inheritance, calculations, and functions.
- A CSS preprocessor is a tool that takes code written in some type of preprocessed language and then converts that code into normal CSS.
- The advantage of a CSS preprocessor is that it can provide additional functionalities that are not available in CSS. One of the best ways to see the power of a CSS preprocessor is with colors.
- For instance, in Figure 2.33, the background color of the .box class, the text color in the <footer> element, the border color of the <fieldset>, and the text color for placeholder text within the <textarea> element, might all be set to #796d6d.
- The trouble with regular CSS is that when a change needs to be made then some type of copy and replace is necessary, which always leaves the possibility that a change might be made to the wrong elements.

- Similarly, it is common for different site elements to have similar CSS formatting, for instance, different boxes to have the same padding. Again, in normal CSS, one has to use copy and paste to create that uniformity.

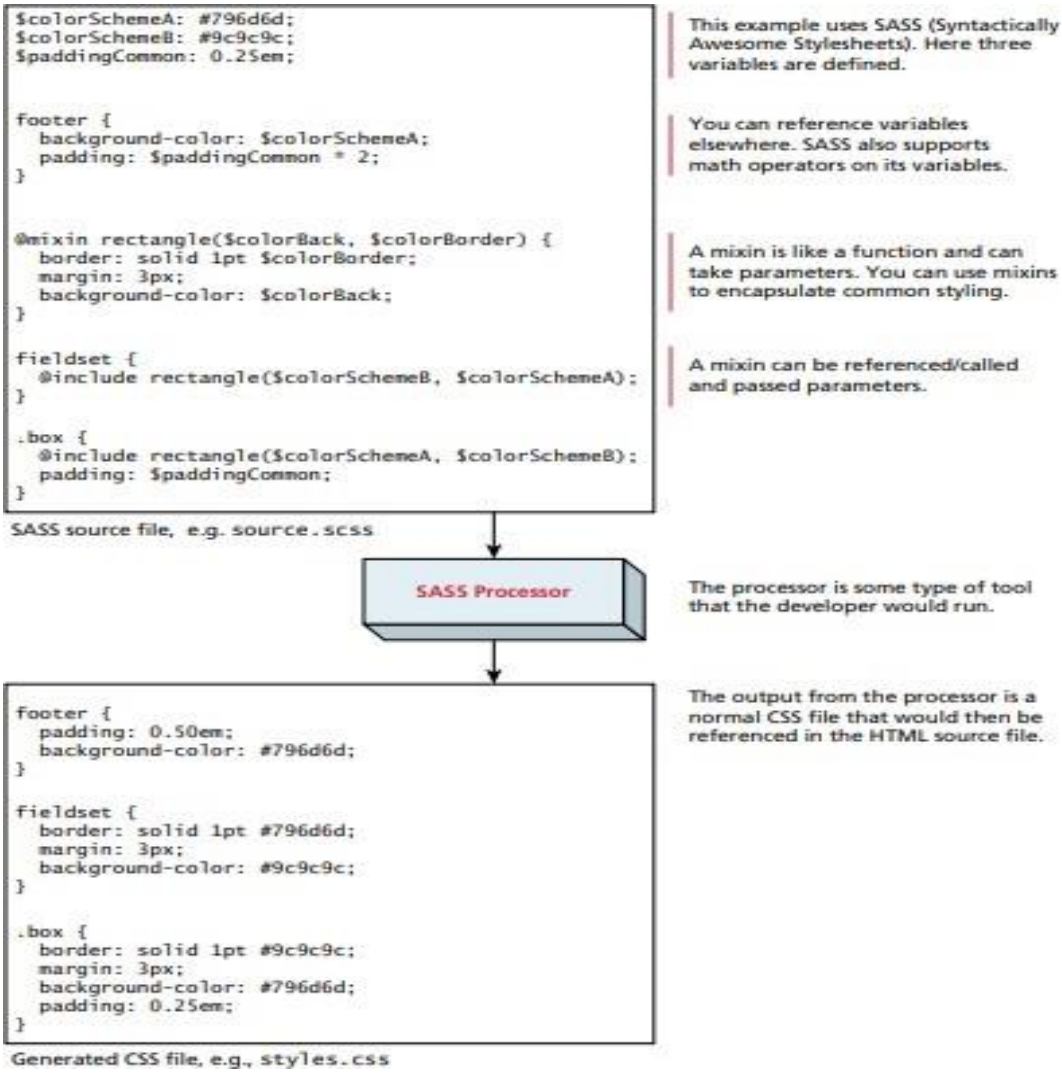


Figure 2.33: Using a CSS preprocessor

MODULE 3

Chapter 1: JavaScript: Client-Side Scripting

1. What is JavaScript and What can it do?
2. JavaScript Design Principles
3. Where does JavaScript Go?
4. Syntax
5. JavaScript Objects
6. The Document Object Model (DOM)
7. JavaScript Events
8. Forms

What Is JavaScript and What Can It Do?

- JavaScript is an object-oriented, dynamically typed, scripting language.
- JavaScript and Java are vastly different programming languages with different uses.
- Java is a full-fledged compiled, object oriented language, popular for its ability to run on any platform with a Java Virtual Machine installed.
- JavaScript is one of the world's most popular languages, with fewer of the object-oriented features of Java, and runs directly inside the browser, without the need for the JVM.
- JavaScript is object oriented, almost everything in the language is an object. **Ex:** variables are objects in that they have constructors, properties, and methods. JavaScript is dynamically typed in that variables can be easily converted from one data type to another.
- In a programming language such as Java, variables are statically typed, in that the data type of a variable is defined by the programmer (e.g., int abc) and enforced by the compiler. With JavaScript, the type of data a variable can hold is assigned at runtime and can change during runtime as well.

Client-Side Scripting

- It refers to the client machine (i.e., the browser) running code locally rather than relying on the server to execute code and return the result.

- There are many client-side languages like Flash, VBScript, Java, and JavaScript. Some of these technologies only work in certain browsers, while others require plug-ins to function.

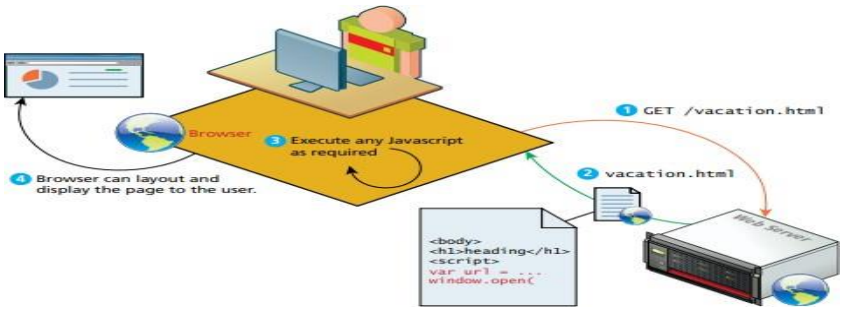


Figure 1.1: Downloading and executing a client-side JavaScript script

Advantages of client-side scripting:

- Processing can be offloaded from the server to client machines, there by reducing the load on the server.
- The browser can respond more rapidly to user events than a request to a remote server ever could, which improves the user experience.
- JavaScript can interact with the downloaded HTML in a way that the server cannot, creating a user experience more like desktop software than simple HTML ever could.

Disadvantages of client-side scripting: are mostly related to how programmers use JavaScript in their applications. Some of these include:

- There is no guarantee that the client has JavaScript enabled, meaning any required functionality must be housed on the server, despite the possibility that it could be offloaded.
- The idiosyncrasies between various browsers and operating systems make it difficult to test for all potential client configurations. What works in one browser, may generate an error in another.
- JavaScript-heavy web applications can be complicated to debug and maintain. JavaScript has often been used through inline HTML hooks that are embedded in to the HTML of a webpage.
- There are two other note worthy client-side approaches to web programming.

Adobe Flash

- It is a vector based drawing and animation program, a video file format, and a software platform that has its own JavaScript-like programming language called **Action Script**.

- Flash is often used for animated advertisements and online games, and can also be used to construct web interfaces.
- It is worth understanding how Flash works in the browser. Flash objects (not videos) are in a format called SWF (Shock wave Flash) and are included within an HTML document via the <object> tag. The SWF file is then downloaded by the browser and then the browser delegates control to a plug-into execute the Flash file, as shown in Figure 1.2.

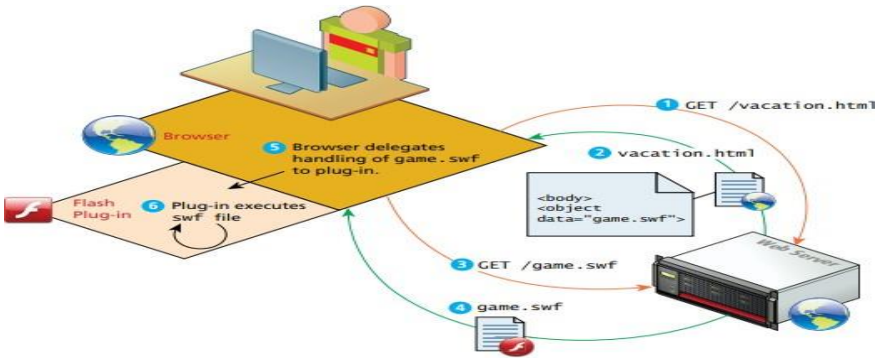


Figure 1.2: Adobe Flash

- A **browser plug-in** is a software add-on that extends the functionality and capabilities of the browser by allowing it to view and process different types of web content. Browser plug-in is different than a **browser extension**.

Java Applets

- An **applet** is a term that refers to a small application that performs a relatively small task.
- Java applets are written using the Java programming language and are separate objects that are included with in an HTML document via the <applet> tag, downloaded, and then passed on to a Java plug-in.
- This plug-in then passes on the execution of the applet outside the browser to the Java Runtime Environment (JRE) that is installed on the client's machine.

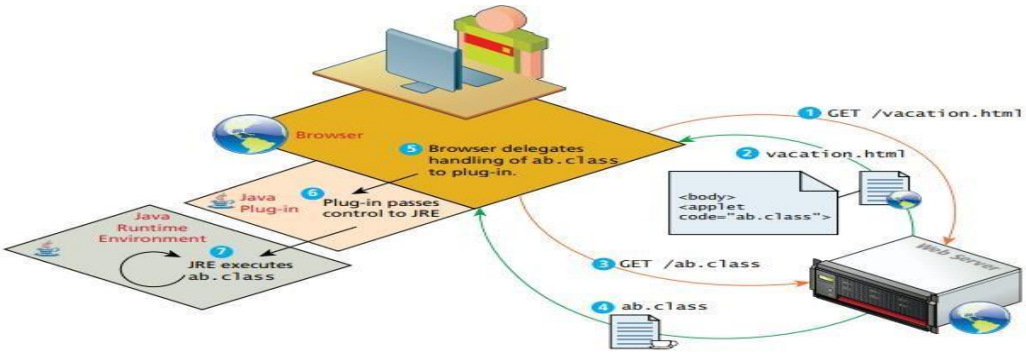


Figure 1.3: Java applets

- Both Flash plug-ins and Java applets are losing support by major players for a number of reasons.
    - Java applets require the JVM be installed and up to date.
    - Flash and Java applets also require frequent updates, which can annoy the user and present security risks.
- With the universal adoption of JavaScript and HTML5, JavaScript remains the most dynamic and important client-side scripting language for the modern web developer.

JavaScript's History and Uses

- JavaScript was introduced by Netscape in their Navigator browser back in 1996. It originally was called Live Script.
- Java Script is in fact an implementation of a standardized scripting language called **ECMA Script**.
- Internet Explorer (IE) at first did not support Java Script, but instead had its own browser-based scripting language (VB Script).
- While IE now does support JavaScript, Microsoft sometimes refers to it as Jscript, primarily for trademark reasons (Oracle currently owns the trademark for JavaScript).

Common uses of java script:

- Graphic roll-overs (that is, swapping one image for another when the user hovered the mouse over an image),
- pop-up alert messages
- scrolling text in the status bar
- opening new browser windows and
- pre-validating user data in online forms.

- AJAX sites that Java Script became a much more important part of web development.
- **AJAX** is both an acronym as well as a general term.
    - As an acronym it means Asynchronous JavaScript and XML, which was accurate for sometime; but since XML is no longer always the data format for data transport in AJAX sites, the acronym meaning is becoming less and less accurate.
    - As a general term, AJAX refers to a style of website development that makes use of JavaScript to create more responsive user experiences.
- Figure1.4 illustrates the processing flow for a page that requires updates based on user input using the normal synchronous non-AJAX page request-response loop.

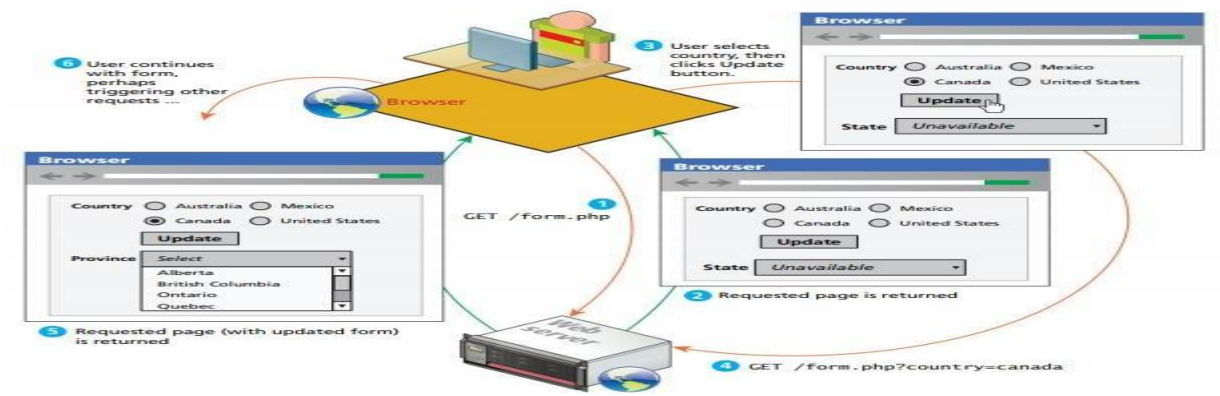


Figure 1.4: Normal HTTP request-response loop

- AJAX provides web authors with away to avoid the visual and temporal deficiencies Of normal HTTP interactions. With AJAX webpages, it is possible to update sections of a page by making special requests of the server in the background.
- This type of AJAX development can be difficult, s o the other key development in the history of JavaScript has made the creation of **JavaScript frameworks**, such as jQuery, Prototype, ASP.NET AJAX, and Moo Tools.
- These JavaScript frameworks reduce the amount of JavaScript code required to perform typical AJAX tasks.
- MVC JavaScript frameworks such as Angular JS, Backbone, and Knock out have gained a lot of interest from developers using a software engineering, namely the separation of the model (data representation) from the view (presentation of data) design pattern.

JavaScript Design Principles

- JavaScript principles increases the quality and reusability of the code while making it easier to understand, and hence have more maintainability.

Layers

- In object-oriented programming, a software **layer** is a way of conceptually grouping programming classes that have similar functionality and dependencies. Common software design layer names include:
  - **Presentation layer.** Classes focused on the user interface.
  - **Business layer.** Classes that model real-world entities, such as customers, products, and sales.
  - **Data layer.** Classes that handle the interaction with the data sources.

- Figure 1.5 illustrates the idea of JavaScript layers.

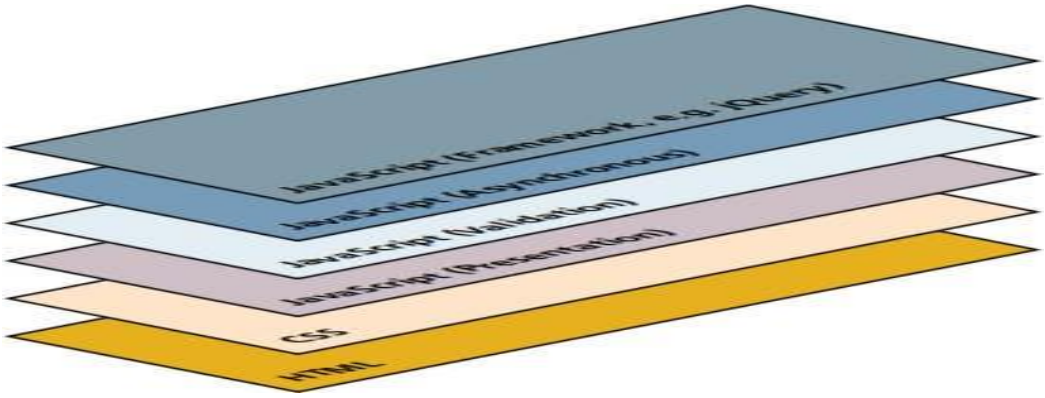


Figure 1.5: JavaScript layers

Presentation Layer

- This type of programming focuses on the display of information. JavaScript can alter the HTML of a page, which results in a change, visible to the user.
- This layer enables creating, hiding, and showing divs, using tabs to show multiple views, or having arrows to page through result sets.
- This layer is most closely related to the user experience and the most visible to the end user.

Validation Layer

- JavaScript can be also used to validate logical aspects of the user's experience. For example, validating a form to make sure the email entered is valid before sending it.
- It is often used in conjunction with the presentation layer, where a message to the presentation layer highlights bad fields.
- Both layers exist on the client machine, although the intention is to pre validate forms before making transmissions back to the server.

Asynchronous Layers

- JavaScript operates in asynchronous manner where a request sent to the server requires a response before the next lines of code can be executed.
- During the wait between request and response the browser sits in a loading state and only updates upon receiving the response. In contrast, an asynchronous layer can route requests to the server in the background.
- The JavaScript sends the HTTP requests to the server, but while waiting for the response, the rest of the application functions normally, and the browser isn't in a loading state.

- When the response arrives JavaScript will update a portion of the page. Asynchronous layers are considered advanced versions of the presentation and validation layers above.

Users without JavaScript

A client may not have JavaScript because they are a web crawler, have a browser plug-in, are using a text browser, or are visually impaired.

- **Web crawler**. A web crawler is a client running on behalf of a search engine to download our site, so that it can eventually be featured in their search results. These automated software agents do not interpret JavaScript, since it is costly, and the crawler cannot see the enhanced look anyway.
- **Browser plug-in**. A browser plug-in is a piece of software that works with in the browser, that might interfere with JavaScript. Many malicious sites use JavaScript to compromise a user's computer, and many ad networks deploy advertisements using JavaScript.
- **Text-based client**. Some clients are using a text-based browser. Text-based browsers are widely deployed on web servers, which are often accessed using a command-line interface.
- **Visually disabled client**. A visually disabled client will use special web browsing software to read the contents of a web page out loud to them.

The <NoScript> Tag

- Any text between the opening and closing tags will only be displayed to users without the ability to load JavaScript. It is often used to prompt users to enable JavaScript, but can also be used to show additional text to search engines.
- we should create websites with all the basic functionality enabled using regular HTML. For majority of users with JavaScript enabled we can then enhance the basic layout using JavaScript.
- This approach of adding functional replacements for those without JavaScript is also referred to as fail-safe design, which is a phrase with a meaning beyond web development. It means that when a plan fails, then the system's design will still work.

Graceful Degradation and Progressive Enhancement

- The principle of fail-safe design can still apply even to browsers that have enabled JavaScript.

- G **raceful degradation**
  - With this strategy we develop our site for the abilities of current browsers. For those users who are not using current browsers, we might provide an alternate site or pages for those using older browsers that lack the JavaScript used on the main site.
  - Figure1.6 (LEFT) illustrates the idea of graceful degradation.
- **Progressive enhancement**
  - which takes the opposite approach to the problem. In this case, the developer creates the site using CSS, JavaScript, and HTML features that are supported by all browsers of a certain age or newer.
  - Figure1.6 (RIGHT) illustrates the idea of progressive enhancement.

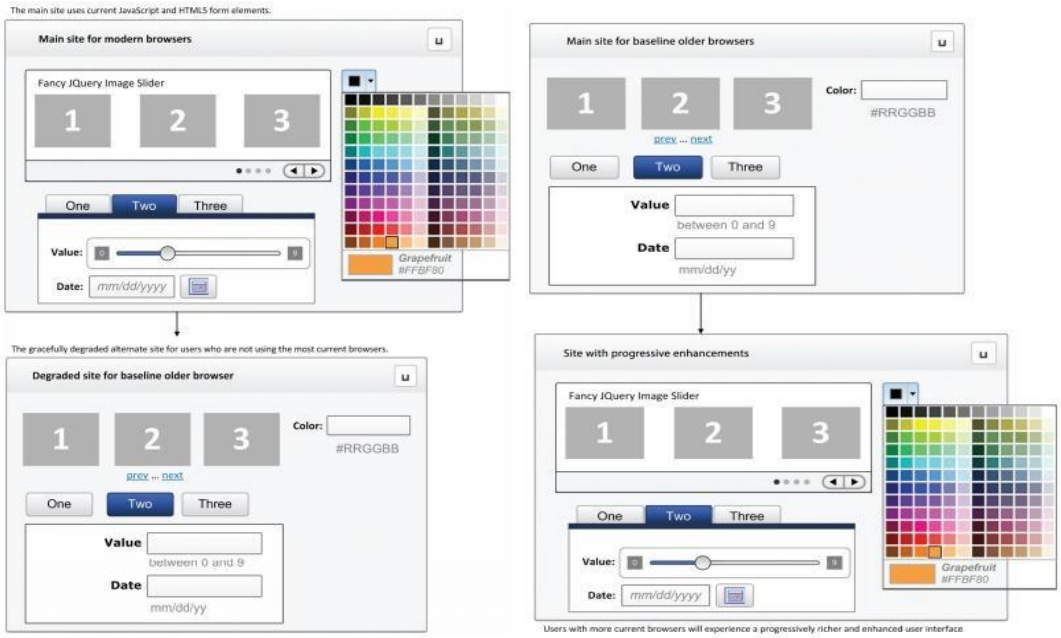


Figure 1.6: Examples of graceful degradation(left) and progressive enhancement(right)

Where Does JavaScript Go?

- JavaScript can be linked to an HTML page in a number of ways. Just as CSS styles can be **inline**, **embedded**, or **external**, but external is the preferred method for cleanliness and ease of maintenance.

Inline JavaScript

- Inline JavaScript includes JavaScript code directly within certain HTML attributes, such as that shown in Listing1.1.
- The same is true with JavaScript. In fact, inline JavaScript is much worse than inline

CSS. Inline JavaScript is a real maintenance nightmare, requiring maintainers to scan Through almost every line of HTML looking for your inline JavaScript.

```
<a href="JavaScript:OpenWindow();">more info</a>
<input type="button" onclick="alert('Are you sure?');" />
```

Listing 1.1: Inline JavaScript example

```
<script type="text/javascript">
/* A JavaScript Comment */
alert ("Hello World!");
</script>
```

Listing 1.2: embedded JavaScript example

Embedded JavaScript

- Embedded JavaScript includes JavaScript code within a <script> element as shown in Listing1.2.Like its equivalent in CSS, embedded JavaScript is okay for quick testing and for learning scenarios, but is frowned upon for normal real world pages. Like with inline JavaScript, embedded scripts can be difficult to maintain

External JavaScript

- JavaScript supports this separation by allowing links to an external file that contains the JavaScript.
- This is the recommended way of including JavaScript scripts in our HTML pages. By convention, JavaScript external files have the extension.js.
- These external files typically contain function definitions, data definitions, and other blocks of JavaScript code.
- In Listing1.3, the link to the external JavaScript file is placed with in the <head> element, a s same as to external CSS files. While this is convention, it is in fact possible to place these links anywhere within the <body> element.
- We certainly recommend placing them either in the <head> element or the very bottom of the <body> element.
- The argument for placing external scripts at the bottom of the <body> has to do with performance. A JavaScript file has to be loaded completely before the browser can begin any other downloads (including images).

```
<head>
  <script type="text/JavaScript" src="greeting.js">
  </script>
</head>
```

Listing 1.3: External JavaScript example

Advanced Inclusion of JavaScript

- Imagine for a moment a user with a browser that has JavaScript disabled.
- When downloading a page, if the JavaScript scripts are embedded in the page, they must download those scripts in their entirety, despite being unable to process them.

Syntax

- Since it's a light weight scripting language.
- JavaScript has some features (such as dynamic typing) that are especially helpful to the novice programmer.
- **Features:**
  - Everything is type sensitive, including function, class, and variable names.
  - The scope of variables in blocks is not supported. This means variables declared inside a loop may be accessible outside of the loop, counter to what one would expect.
  - There is a === operator, which tests not only for equality but type equivalence.
  - Null and undefined are two distinctly different states for a variable.
  - Semicolons are not required, but are permitted (and encouraged).
  - There is no integer type, only number, which means floating-point rounding errors are prevalent even with values intended to be integers.

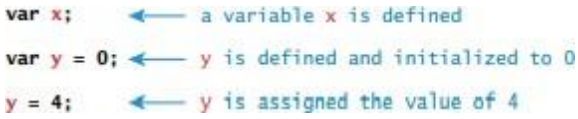


Figure 1.7: Variable declaration and assignment

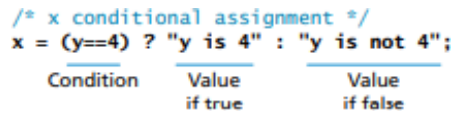


Figure 1.8: Conditional assignment

Variables

- **Variables** in JavaScript are **dynamically typed**, meaning a variable can be an integer, And then later a string, then later an object, if so desired.
- This simplifies variable declarations, so that we do not require the familiar type fields like int, char, and String. Instead, to declare a variables, we use the var keyword, the name, and a semicolon as shown in Figure 1.7.
- If we specify no value, then (being type less) the default value is undefined.
- **Assignment** can happen at declaration-time by appending the value to the declaration, or at runtime with a simple right-to-left assignment.
- The **conditional assignment** operator, shown in Figure 1.8, can also be used to assign based on condition, although its use is sometimes discouraged.

Comparison Operators

- The core of any programming language is the ability to distill things down to Boolean statements where something is either true or false. JavaScript is no exception and comes equipped with a number of operators to compare two values, listed in Table 1.1.

Operator	Description	Matches (x=9)
==	Equals	(x==9) is true (x=="9") is true
===	Exactly equals, including type	(x==="9") is false (x===9) is true
< , >	Less than, greater than	(x<5) is false
<= , >=	Less than or equal, greater than or equal	(x<=9) is true
!=	Not equal	(4!=x) is true
!==	Not equal in either value or type	(x!=="9") is true (x!==9) is false

Table 1.1: Comparison Operators

- These comparison operators are used in conditional, loop, and assignment statements.

Logical Operators

- Comparison operators are useful, but without being able to combine several together, their usefulness would be severely limited.
- Therefore, like most languages JavaScript includes Boolean operators, which allow us to build complicated expressions.
- The Boolean operators and, or, and not and their truth tables are listed in Table 1.2. Syntactically they are represented with && (and), ||(or), and! (not).

A	B	A && B
T	T	T
T	F	F
F	T	F
F	F	F

AND Truth Table

A	B	A || B
T	T	T
T	F	T
F	T	T
F	F	F

OR Truth Table

A	! A
T	F
F	T

NOT Truth Table

Table 1.2: AND, OR, and NOT Truth Tables

Conditionals

- JavaScript's syntax is almost identical to that of PHP, Java, or C when it comes to conditional structures such as if and ifelse statements.
- In this syntax the condition to test is contained within() brackets with the body contained in{} blocks.
- Optional elseif statements can follow, with an else ending the branch. Listing 1.4 uses a conditional to set a greeting variable, depending on the hour of the day.

```
var hourOfDay;   // var to hold hour of day, set it later...
var greeting;    // var to hold the greeting message.
if (hourOfDay > 4 && hourOfDay < 12){
   // if statement with condition
   greeting =  "Good Morning";
}
else if (hourOfDay >= 12 && hourOfDay < 20){
   // optional else if
   greeting =  "Good Afternoon";
}
else{ // optional else branch
   greeting = "Good Evening";
}
```

Listing 1.4: Conditional statement setting a variable based on the hour of the day

Loops

Like conditionals, loops use the() and{} blocks to define the condition and the body of the loop.

While Loops

- The most basic loop is the while loop, which loops until the condition is not met.
- Loops normally initialize a **loop control variable** before the loop, use it in the condition, and modify it with in the loop.
- One must be sure that the variables that make up the condition are updated inside the loop (or elsewhere) to avoid an infinite loop!

```
var i=0;
while(i < 10){
  //do something with i
  i++;
}
```

For Loops

- A **for loop** combines the common components of a loop: initialization, condition, and post-loop operation into one statement.
- This statement begins with the for keyword and has the components placed between() brackets, semicolon(;) separated.

```
for (var i = 0; i < 10; i++){
  //do something with i
}
```

Functions

- **Functions** are the building block for modular code in JavaScript, and are even used to build **pseudo-classes**.
- They are defined by using the reserved word function and then the function name and (optional) parameters.
- Since JavaScript is dynamically typed, functions do not require a return type, nor do the parameters require type. Therefore a function to raise x to the yth power might be defined as:
- With new programmers there is often confusion between defining a function and calling the function. Remember that when actually using the keyword function, we are defining what the function does.

```
function power(x,y){
   var pow=1;
   for (var i=0;i<y;i++){
      pow = pow*x;
   }
   return pow;
}
And called as
power(2,10);
```

Alert

- The alert() function makes the browser show a pop-up to the user, with whatever is passed being the message displayed.
- **Ex:**

```
alert ( "Good Morning" );
```

- The pop-up may appear different to each user depending on their browser configuration. What is universal is that the pop-up obscures the underlying webpage, and no actions can be done until the pop-up is dismissed.
- Alerts are not used in production code, but area useful tool for debugging and illustration purposes.

Errors Using Try and Catch

- When the browser's JavaScript engine encounters an error, it will *throw* an **exception**.
- These exceptions interrupt the regular, sequential execution of the program and can stop the JavaScript engine altogether.
- However, we can optionally catch these errors preventing disruption of the program using the **try–catch block** as shown in Listing 1.5.

```
try {
  nonexistantfunction("hello");
}
catch(err) {
  alert("An exception was caught:" + err);
}
```

Listing 1.5: Try-catch statement

```
try {
   var x = -1;
   if (x<0)
      throw "smallerthan0Error";
}
catch(err){
   alert (err + "was thrown");
}
```

Listing 1.6: Throwing a user-defined exception

Throwing Your Own Exceptions

- Although try-catch can be used exclusively to catch built-in JavaScript errors, it can also be used by our programs, to throw our own messages.
- The throw keyword stops normal sequential execution, just like the built-in exceptions as shown in Listing1.6 demonstrates the throwing of a user-defined exception as a string.
- The general consensus in software development is that try-catch and throw statements should be used for *abnormal* or *exceptional* cases in our program.
- They should not be used as a normal way of controlling flow, although no formal mechanism exists to enforce that idea.
- We will generally avoid try-catch statements in our code unless illustrative of some particular point.
- In reality any object can be thrown, although in practice a string usually suffices.

- It should be noted that throwing an exception disrupts the sequential execution of a program.
- That is, when the exception is thrown all subsequent code is not executed until the catch statement is reached.

JavaScript Objects

- Objects can have **constructors**, **properties**, and **methods** associated with them, and are used very much like objects in other object-oriented languages.
- There are objects that are included in the JavaScript language; we can also define our own kind of objects.

Constructors

- Normally to create a new object we use the **new** keyword, the classname, and () brackets with *n* optional parameters inside, comma delimited as follows.

```
var someObject = new ObjectName(parameter 1,param 2,..., parameter n);
```

- For some classes, shortcut constructors are defined, which can be confusing if we are not aware of them. **Ex,** a String object can be defined with the shortcut.

```
var greeting = "Good Morning";
Instead of the formal definition
var greeting = new String("Good Morning");
```

Properties

- Each object might have properties that can be accessed, depending on its definition. When a property exists, it can be accessed using **dot notation** where a dot between the instance name and the property references that property.

```
alert(someObject.property); //show someObject.property to the user
```

Methods

- Objects can also have methods, which are functions associated with an instance of an object.
- These methods are called using the same dot notation as for properties, but instead of accessing a variable, we are calling a method.

```
someObject.doSomething();
```

- Methods may produce different output depending on the object they are associated with because they can utilize the internal properties of the object.

Objects Included in JavaScript

- A number of useful objects are included with JavaScript.
- These include Array, Boolean, Date, Math, String, and others. In addition to these, Java Script can also access Document Object Model (DOM) objects that correspond to the content of a page's HTML.
- These DOM objects let JavaScript code access and modify HTML and CSS properties of a page dynamically.

Arrays

- Arrays are one of the most used data structures, and they have been included in JavaScript as well.
- The class is defined to behave more like a linked list in that it can be resized dynamically, but the implementation is browser specific, meaning the efficiency of insert and delete operations is unknown.
- Arrays will be the first objects we will examine. Objects can be created using the new syntax and calling the object constructor. The following code creates a new, empty array named greetings:

```
var greetings = new Array();
```

- To initialize the array with values, the variable declaration would look like the following:

```
var greetings = new Array("Good Morning", "Good Afternoon");
```

or, using the square bracket notation:

```
var greetings = ["Good Morning", "Good Afternoon"];
```

Accessing and Traversing an Array

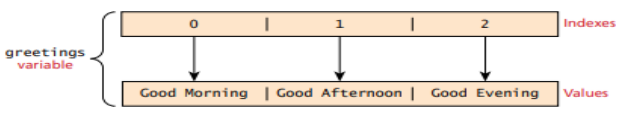
- To access an element in the array we use the familiar square bracket notation from Java and C-style languages, with the index we wish to access inside the brackets.

```
alert ( greetings[0] );
```

- One of the most common actions on an array is to traverse through the items sequentially.
- The following for loop quickly loops through an array, accessing the ith element each time using the Array object's length property to determine the maximum valid index. It will alert "Good Morning" and "Good Afternoon" to the user.

```
for (var i = 0; i < greetings.length; i++){
    alert(greetings[i]);

}
```



Modifying an Array

- To add an item to an existing array, you can use the push method.

```
greetings.push("Good Evening");
```

- The pop method can be used to remove an item from the back of an array.
- Additional methods that modify arrays include concat(), slice(), join(), reverse(), shift(), and sort().

Math

- The **Math class** allows one to access common mathematic functions and common values quickly in one place.
- This static class contains methods such as max(), min(), pow(), sqrt(), and exp(), and trigonometric functions such as sin(), cos(), and arctan().
- In addition, many mathematical constants are defined such as PI, E(Euler's number), SQRT2, and some others as shown in Listing1.7.

```
Math.PI            // 3.141592657
Math.sqrt(4);      // square root of 4 is 2.
Math.random();     // random number between 0 and 1
```

Listing 1.7: Some constants and functions in the Math object

String

- The **String class** has already been used without us even knowing it.
- While one can use the new syntax to create a String object, it can also be defined using quotes as follows:

```
var greet = new String("Good");    // long form constructor
var greet = "Good";                // shortcut constructor
```

- Length property: `alert (greet.length); // will display "4"`
- Another common way to use strings is to concatenate them together. Since this is so common, the + operator has been overridden to allow for concatenation in place.

```
var str = greet.concat("Morning");    // Long form concatenation
var str = greet + "Morning";          // + operator concatenation
```

- Many other useful methods exist within the String class, such as
  - accessing as single character using char At(), or
  - searching for one using index Of().
  - Strings allow splitting a string into an array,
  - searching and matching with split(),
  - search(), and match() methods.

Date

- The Date class is yet another helpful included object we should be aware of.
- It allows us to quickly calculate the current date or create date objects for particular dates.
- To display today's date as a string, we would simply create a new object and use the toString() method.

```
var d = new Date();
// This outputs Today is Mon Nov 12 2012 15:40:19  GMT-0700
alert ("Today is "+ d.toString());
```

Window Object

- The window object in JavaScript corresponds to the browser itself.
- Through it , we can access the current page's URL, the browser's history, and what's being displayed in the status bar, as well as opening new browser windows.
- In fact, the alert() function mentioned earlier is actually a method of the window object.

The Document Object Model(DOM)

- Java Script is almost always used to interact with the HTML document in which it is contained. As such, there needs to be some way of programmatically accessing the elements and attributes within the HTML.
- This is accomplished through a programming interface (API) called the **Document Object Model (DOM)**.
- The tree structure (as shown in below figure 1.9) is formally called the **DOM Tree** with the root, or topmost object called the **Document Root**.

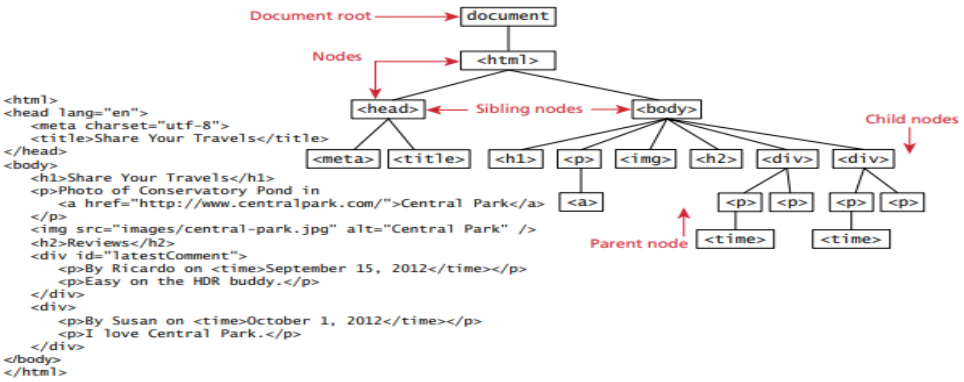


Figure 1.9: DOM tree

Nodes

- In the DOM, each element within the HTML document is called a **node**.
- If the DOM is a tree, then each node is an individual branch.

- There are element nodes, text nodes, and attribute nodes, as shown in Figure 1.10.
- All nodes in the DOM share a common set of properties and methods.
- Thus, most of the tasks that we typically perform in JavaScript involve finding a node, and then accessing or modifying it via those properties and methods. The most important of these are shown in Table 1.3.

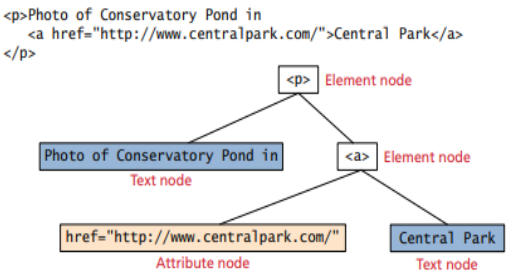


Figure 1.10: DOM nodes

Property	Description
attributes	Collection of node attributes
childNodes	A NodeList of child nodes for this node
firstChild	First child node of this node
lastChild	Last child of this node
nextSibling	Next sibling node for this node
nodeName	Name of the node
nodeType	Type of the node
nodeValue	Value of the node
parentNode	Parent node for this node
previousSibling	Previous sibling node for this node.

Table 1.3: Node Object Properties

Document Object

- The **DOM document object** is the root Java Script object representing the entire HTML document.
- It contains some properties and methods that we will use extensively in our development and is globally accessible as document.
- The attributes of this object include some information about the page including doc type and input Encoding.
- Accessing the properties is done through the dot notation.

```
// specify the doctype, for example html
var a = document.doctype.name;
// specify the page encoding, for example ISO-8859-1
var b = document.inputEncoding;
```

- There are some essential methods (Table 1.4) we will use all the time. They include getElementByTagName() and the indispensable getElementById().

Method	Description
createAttribute()	Creates an attribute node
createElement()	Creates an element node
createTextNode()	Creates a text node
getElementById(id)	Returns the element node whose id attribute matches the passed id parameter
getElementsByTagName(name)	Returns a NodeList of elements whose tag name matches the passed name parameter

Table 1.4: Some Essential Document Object Methods

- While the former method returns an array of DOM nodes (called a Node List) matching the tag, the latter returns a single DOM element that matches the id passed as a parameter as illustrated in Figure 1.11.

Figure1.11: Relationship between getElementByID() and getElementsByTagName()

- The method getElementById() is universally implemented and thus used extensively. The newer querySelector() and querySelectorAll() methods allow us to query for DOM elements much the same way we specify CSS styles.

Element Node Object

- The type of object returned by the method document.getElementById() is an element node object.
- This presents an HTML element in the hierarchy, contained between the opening <> and closing </> tags for this element.
- An element can itself contain more elements. Since IDs must be unique in an HTML document, getElementByID() returns a single node, rather than a set of results which is the case with other selector functions.
- The returned Element Node object has the node properties shown in Table 1.3.It also has a variety of additional properties, the most important of which are shown in Table 1.5.

Property	Description
`className`	The current value for the `class` attribute of this HTML element.
`id`	The current value for the `id` of this element.
`innerHTML`	Represents all the things inside of the tags. This can be read or written to and is the primary way in which we update particular elements using JavaScript.
`style`	The `style` attribute of an element. We can read and modify this property.
`tagName`	The tag name for the element.

Table 1.5: Element Node Properties

Property	Description	Tags
`href`	The href attribute used in a tag to specify a URL to link to.	`a`
`name`	The `name` property is a bookmark to identify this tag. Unlike `id`, which is available to all tags, `name` is limited to certain form-related tags.	`a, input, textarea, form`
`src`	Links to an external URL that should be loaded into the page (as opposed to `href`, which is a link to follow when clicked)	`img, input, iframe, script`
`value`	The value is related to the `value` attribute of input tags. Often the value of an input field is user defined, and we use `value` to get that user input.	`input, textarea, submit`

Table 1.6: HTML DOM Element Properties

- While these properties are available for all HTML elements, there are some HTML Elements that have additional properties that can be accessed. Table 1.6 lists some common additional properties and the HTML tags that have these properties.

Modifying a DOM Element

- The document.write() method is used to create output to the HTML page from JavaScript. While this is certainly valid, it always creates JavaScript at the bottom of the existing HTML page.
- Using the DOM document and HTML DOM element objects, we can do exactly that using the inner HTML property as shown in Listing 1.8 (using the HTML shown in Figure 1.11)

```
var latest = document.getElementById("latestComment");
var oldMessage = latest.innerHTML;
latest.innerHTML = oldMessage + "<p>Updated this div with JS</p>";
```

Listing 1.8: Changing the HTML using innerHTML

A More Verbose Technique

- Although the inner HTML technique works well there is a more verbose technique available to us that builds output using the DOM.
- This more explicit technique has the advantage of ensuring that only valid markup is created, while the inner HTML could output badly formed HTML.
- DOM functions create-Text Node(), remove Child(), and append Child() allow us to modify an element in a more rigorous way as shown in Listing1.9.

```
var latest = document.getElementById("latestComment");
var oldMessage = latest.innerHTML;
var newMessage = oldMessage + "<p>Updated this div with JS</p>";
latest.removeChild(latest.firstChild);
latest.appendChild(document.createTextNode(newMessage));
```

Listing 1.9: Changing the HTML using createTextNode( ) and appendChild( )

Changing an Element's Style

- We can add or remove any style using the style or class Name property of the Element node, which is something that we might want to do to dynamically change the appearance of an element.
- Its usage is shown below to change a node's background color and add a three-pixel border.

```
var commentTag = document.getElementById("specificTag");
commentTag.style.backgroundColour = "#FFFF00";
commentTag.style.borderWidth="3px";
```

- With knowledge of CSS attributes we can easily change any style attribute.
- The style property is itself an object, specifically a CSS Style Declaration type, which includes all the CSS attributes as properties and computes the current style from inline, external, and embedded styles.

- The class Name property is normally a better choice, because it allows the styles to be created outside the code, and thus be better accessible to designers.
- Using this model we would change the background color by having two styles defined, and changing them in Java Script code.

```
var commentTag = document.getElementById("specificTag");
commentTag.className = "someClassName";
```

- HTML5 introduces the classList element, which allows you to add, remove, or toggle a CSS class on an element. You could add a class with

```
label.classList.addClass("someClassName");
```

Additional Properties

- In addition to the global properties present in all tags, there are additional methods available when dealing with certain tags. Table 1.6 lists a few common ones.

  To get the password out of the following input field and alert the user.

```
<input type='password' name='pw' id='pw' />
```

- It should be obvious show getting the src or href properties out of appropriate tags could also be done. We leave it as an exercise to the reader.

```
var pass = document.getelementById("pw");
alert (pass.value);
```

JavaScript Events

- At the core of all JavaScript programming is the concept of an **event**.
- A JavaScript event is an action that can be detected by JavaScript. Many of them are initiated by user actions but some are generated by the browser itself.
- We say then that an event is *triggered* and then it can be *caught* by JavaScript functions, which then do something in response.
- As more powerful frameworks were developed, and website design and best practices were refined, this original mechanism was supplanted by the **listener** approach.

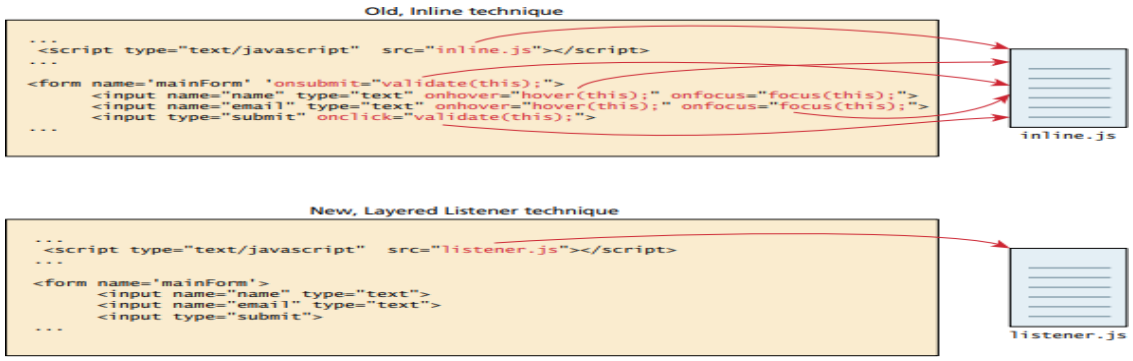


Figure 1.12: Inline hooks versus the Layered Listener technique

- A visual comparison of the old and new technique is shown in Figure 1.12. Note how the old method weaves the Java Script right inside the HTML, while the listener technique has removed JavaScript from the markup, resulting in cleaner, easier to maintain HTML code.

Inline Event Handler Approach

- JavaScript events allow the programmer to react to user interactions.
- In early web development, it made sense to weave code and HTML together and to this day, inline Java Script calls are intuitive.
- **Ex**, if we wanted an alert to pop-up, when clicking a we might program:

```
<div id="example1" onclick="alert('hello')">Click for pop-up</div>
```

- In this example the HTML attribute **onclick** is used to attach a handler to that event.
- When the user clicks the , the event is triggered and the alert is executed.

Listener Approach

- The problem with the inline handler approach is that it does not make use of layers; that is, it does not separate content from behavior.

```
var greetingBox = document.getElementById('example1');
greetingBox.onclick = alert('Good Morning');
```

Listing 1.10: The "old"(registering a listener)

```
var greetingBox = document.getElementById('example1');
greetingBox.addEventListener('click', alert('Good Morning'));
greetingBox.addEventListener('mouseOut', alert('Goodbye'));

// IE 8
greetingBox.attachEvent('click', alert('Good Morning'));
```

Listing 1.11: The "new" (registering listeners)

- The approach shown in Listing 1.10 is widely supported by all browsers. The first line in the listing creates a temporary variable for the HTML element that will trigger the event.
- The next line attaches the element's on click event to the event handler, which invokes the Java Script alert() method.
- The main **advantage** of this approach is that this code can be written anywhere, including an external file that helps *uncouple* the HTML from the JavaScript.
- The one **limitation** with this approach (and the inline approach) is that only one handler can respond to any given element event.
- The use of add EventListener() shown in Listing 1.11 was introduced in DOM Version2, and as such is unfortunately not supported by IE 8 or earlier.
- This approach has all the other advantages of the approach shown in Listing 1.10, And has the additional advantage that multiple handlers can be assigned to a single object's event.

- The examples in Listing 1.10 and Listing 1.11 simply used the built-in JavaScript alert() function.
- What if we wanted to do something more elaborate when an event is triggered? In such a case, the behavior would have to been capsulated within a function, as shown in Listing 1.12.

```
function displayTheDate() {
   var d = new Date();
   alert ("You clicked this on "+ d.toString());
}
var element = document.getElementById('example1');
element.onclick = displayTheDate;

// or using the other approach
element.addEventListener('click',displayTheDate);
```

Listing 1.12: Listening to an event with a function

```
var element = document.getElementById('example1');
element.onclick = function() {
   var d = new Date();
   alert ("You clicked this on " + d.toString());
};
```

Listing 1.13: An event with an anonymous function

- An alternative to that shown in Listing 1.12 is to use an anonymous function(that is, One without a name), as shown in Listing 1.13. This approach is especially common when the event handling function will only ever be used as a listener.

Event Object

- No matter which type of event we encounter, they are all **DOM event objects** and the event handlers associated with them can access and manipulate them.
- Typically we see the events passed to the function handler as a parameter named *e*.

```
function someHandler(e) {
   // e is the event that triggered this handler.
}
```

- These objects have many properties and methods.
    - **Bubbles**.
        - The bubbles property is a Boolean value. If an event's bubble property is set to true then there must be an event handler in place to handle the event or it will bubble up to its parent and trigger an event handler there.
        - If the parent has no handler it continues to bubble up until it hits the document root, and then it goes away, unhandled.
    - **Cancelable**.
        - The Cancelable property is also a Boolean value that indicates whether or not the event can be cancelled.

- If an event is cancelable, then the default action associated with it can be canceled.
- A common example is a user clicking on a link. The default action is to follow the link and load the new page.

- **prevent Default**.
  - A cancelable default action for an event can be stopped using the prevent Default() method as shown in Listing 1.14.

```
function submitButtonClicked(e) {
  if (e.cancelable){
    e. preventDefault();
  }
}
```

Listing 1.14: A function that prevents the default event

  - This is a common practice when we want to send data asynchronously when a form is submitted, **Ex** since the default event of a form submit click is to post to a new URL (which causes the browser to refresh the entire page).

Event Types

- The most obvious event is the click event, but JavaScript and the DOM support several others.
- In actuality there are several classes of event, with several types of event with in each class specified by the W3C.
- The classes are mouse events, keyboard events, form events, and frame events.

Mouse Events

- Mouse events are defined to capture a range of inter actions driven by the mouse. These can be further categorized as mouse click and mouse move events.
- Table 1.7 lists the possible events one can listen for from the mouse. Interestingly, many mouse events can be sent at a time.
- The user could be moving the mouse off one and on to another in the same moment, triggering on mouse on and on mouse out events as well as the on mouse move event.
- The Cancelable and Bubbles properties can be used to handle these complexities.

Event	Description
onclick	The mouse was clicked on an element
ondblclick	The mouse was double clicked on an element
onmousedown	The mouse was pressed down over an element
onmouseup	The mouse was released over an element
onmouseover	The mouse was moved (not clicked) over an element
onmouseout	The mouse was moved off of an element
onmousemove	The mouse was moved while over an element

Table 1.7: Mouse Events in JavaScript

Event	Description
onkeydown	The user is pressing a key (this happens first)
onkeypress	The user presses a key (this happens after onkeydown)
onkeyup	The user releases a key that was down (this happens last)

Table 1.8: Keyboard Events in JavaScript

Keyboard Events

- Keyboard events are often overlooked by novice web developers, but are important tools for power users.
- Table 1.8 lists the possible key board events. These events are most useful within input fields.
- **Ex** validate an email address, or send an asynchronous request for a drop down list of suggestions with each key press.

```
<input type="text" id="keyExample">
```

- The input box above, for example, could be listened to and each key pressed echoed back to the user as an alert as shown in Listing 1.14.

```
document.getElementById("keyExample").onkeydown = function
myFunction(e){
   var keyPressed=e.keyCode;      //get the raw key code
   var character=String.fromCharCode(keyPressed); //convert to string
   alert("Key " + character + " was pressed");
}
```

Listing 1.14: Listener that hears and alerts key presses

Form Events

- Forms are the main means by which user input is collected and transmitted to the server.
- Table 1.9 lists the different form events. The events triggered by forms allow us to do some timely processing in response to user input.
- The most common JavaScript listener for forms is the **onsubmit** event.
- In the code below we listen for that event on a form with id login Form. If the password field (with id pw) is blank, we prevent submitting to the server using prevent Default()and alert the user.

- Otherwise we do nothing, which allows the default event to happen (submitting the form) as shown in Listing 1.15.

Event	Description
onblur	A form element has lost focus (that is, control has moved to a different element), perhaps due to a click or Tab key press.
onchange	Some <input>, <textarea>, or <select> field had their value change. This could mean the user typed something, or selected a new choice.
onfocus	Complementing the onblur event, this is triggered when an element gets focus (the user clicks in the field or tabs to it).
onreset	HTML forms have the ability to be reset. This event is triggered when that happens.
onselect	When the users selects some text. This is often used to try and prevent copy/paste.
onsubmit	When the form is submitted this event is triggered. We can do some prevalidation when the user submits the form in JavaScript before sending the data on to the server.

Table 1.9: Form Events in JavaScript

Event	Description
onabort	An object was stopped from loading
onerror	An object or image did not properly load
onload	When a document or object has been loaded
onresize	The document view was resized
onscroll	The document view was scrolled
onunload	The document has unloaded

Table 1.10: Frame Events in JavaScript

```
document.getElementById("loginForm").onsubmit = function(e){
  var pass = document.getElementById("pw").value;
  if(pass==""){
    alert ("enter a password");
    e.preventDefault();
  }
}
```

Listing 1.15: Catching the onsubmit event and validating a password to not be blank

Frame Events

- Frame events (see Table 1.10) are the events related to the browser frame that contains our webpage.
- The most important event is the onload event, which tells us an object is loaded and therefore ready to work with.
- In fact, every non trivial event listener we write requires that the HTML be fully loaded.
- However, a problem can occur if the Java Script tries to reference a particular in the HTML page that has not yet been loaded.
- If the code attempts to set up a listener on this not-yet-loaded , then an error will be triggered.
- For this reason it is common practice to use the window.onload event to trigger the execution of the rest of the page's scripts.

```
window.onload= function(){
    //all JavaScript initialization here.
}
```

Forms

To illustrate some form-related JavaScript concepts, consider the simple HTML form depicted in Listing 1.16.

```
<form action='login.php' method='post' id='loginForm'>
  <input type='text' name='username' id='username'/>
  <input type='password' name='password' id='password'/>
  <input type='submit'></input>
</form>
```

Listing 1.16: A basic HTML form for a login example

Validating Forms

- Form validation is one of the most common applications of JavaScript.
- Writing code to prevalidate forms on the client side will reduce the number of incorrect submissions, thereby reducing server load.
- Although validation must still happen on the server side (in case JavaScript was circumvented), JavaScript prevalidation is a best practice.
- There are a number of common validation activities including email validation, number validation, and data validation.
- In practice regular expressions are used, and allow for more complete and concise scripts to validate particular fields.
- However, the novice programmer may not be familiar or comfortable using regex, and will often resort to copying a regex from the Internet, without understanding how it works, and therefore, will be unable to determine if It is correct.

Empty Field Validation

- A common application of a client-side validation is to make sure the user entered Something in to a field.
- There's certainly no points ending a request to login if the user name was left blank, so why not prevent the request from working ?
- The way to check for an empty field in JavaScript is to compare a value to both null and the empty string("") to ensure it is not empty, as shown in Listing 1.17 related JavaScript concepts, consider the simple HTML form depicted in Listing 1.16.

```
document.getElementById("loginForm").onsubmit = function(e){
  var fieldValue=document.getElementByID("username").value;
  if(fieldValue==null || fieldValue== ""){
    // the field was empty. Stop form submission
    e.preventDefault();
    // Now tell the user something went wrong
    alert("you must enter a username");
  }
}
```

Listing 1.17: Validation to check for empty fields

```
function isNumeric(n) {
    return !isNaN(parseFloat(n)) && isFinite(n);
}
```

Listing 1.18: A function to test for a numeric value

Number Validation

- Number validation can take many forms. We might be asking users for their age for example, and then allow them to type it rather than select it.
- Unfortunately, no simple functions exist for number validation like one might expect from a full fledged library. Using parse Int(), is NAN(), and is Finite().
- Part of the problem is that JavaScript is dynamically typed, so "2"!==2, but "2"==2. jQuery and a number of programmers have worked extensively on this issue and have come up with the function is Numeric() shown in Listing 1.18.

Note: This function will not parse "European" style numbers with commas (i.e., 12.00 vs.12,00).

Submitting Forms

- Submitting a form using JavaScript requires having a node variable for the form element.
- Once the variable, say, form Example is acquired, one can simply call the submit() method:

```
var formExample = document.getElementById("loginForm");
formExample.submit();
```

- This is often done in conjunction with calling prevent Default() on the onsubmit event.
- This can be used to submit a form when the user did not click the submit button, or to submit forms with no submit buttons at all (say we want to use an image instead).
- Also, this can allow JavaScript to do some processing before submitting a form, perhaps updating some values before transmitting.
- It is possible to submit a form multiple times by clicking buttons quickly, which means our server-side scripts should be designed to handle that eventuality.
- Clicking a submit button twice on a form should not result in a double order, double email, or double account creation, so keep that in mind as we design your applications.

Chapter 2: Introduction to Server-Side Development with PHP

1. What is Server-Side Development
2. A Web Server's Responsibilities
3. Quick Tour of PHP
4. Program Control
5. Functions

What Is Server-Side Development?

- The basic hosting of our files is achieved through a web server.
- Server-side development is much more than web hosting: it involves the use of a programming technology like PHP or ASP. NET to create scripts that dynamically generate content.
- It is important to remember that when developing server-side scripts, we are writing software, just like a C or Java programmer would do, with the major distinction that your software runs on a web server and uses the HTTP request response loop for most interactions with the clients.

Comparing Client and Server Scripts

- Figure 2.1 illustrates how client and server scripts differ.

Client-side Scripts	Server side Scripts
Client-side scripts are executed on the client browser.	Server-side scripts are executed on the web server.
Java Script code is downloaded to the client and is executed there.	The server sends the JavaScript, but you have no guarantee that the script will even execute.
The clients never get to see the code, just the HTML output from the script.	Server-side source code remains hidden from the client as it is processed on the server.
Client-side scripts cannot make use of the resources.	Server-side scripts can make use of resources available at the server.

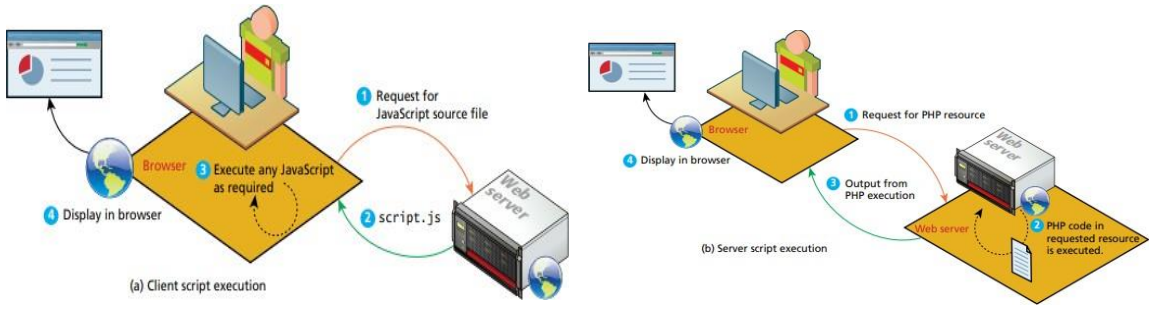


Figure 2.1: Comparison of (a) client script execution and (b) server script execution

Server-Side Script Resources

- A server-side script can access any resources made available to it by the server like data storage resources, web services and software applications.
- **Data Storage** The most commonly used resource is **data storage**, often in the form of a connection to a database management system.
  - A **database management system(DBMS)** is a software system for storing, retrieving, and organizing large amounts of data.
- **Web services** use the HTTP protocol to return XML or other data formats and are often used to extend the functionality of a website.
- **Additional software** that can be installed on a server or accessed via a network connection. Using other software like server applications can send and receive email, access user authentication services, and use network accessible storage.

Comparing Server-Side Technologies

- **ASP (Active Server Pages).**
  - This was Microsoft's first server-side technology (also called ASP Classic).
  - Like PHP, ASP code (using the VBScript programming language) can be embedded within the HTML; it supported classes and some object-oriented features. Most developers did not make use of these features.
  - ASP programming code is interpreted at run time, hence it can be slow in comparison to other technologies.
- **ASP.NET.**
  - This replaced Microsoft's older ASP technology.
  - ASP.NET is part of Microsoft's .NET Framework and can use any .NET programming language (though C# is the most commonly used).

- It also uses special markup called web server controls that encapsulate common web functionality such as database-driven lists, form validation, and user registration wizards.
- A recent extension called ASP.NET MVC makes use of the Model-View Controller design pattern.
- ASP. NET pages are compiled into an intermediary file format called MSIL that is analogous to Java's byte-code. ASP.NET then uses a JIT (Just-In-Time) compiler to compile the MSIL into machine executable code so its performance can be excellent.

- **JSP (Java Server Pages).**
  - JSP uses Java as its programming language and like ASP.NET it uses an explicit object-oriented approach and is used in large enterprise web systems and is integrated into the J2EE environment.
  - It also uses a JIT compiler for fast execution time and is cross-platform.
- **Node.js.**
  - This is a more recent server environment that uses JavaScript on the server side, thus allowing developers already familiar with JavaScript to use just a single language for both client-side and server-side development.
  - Unlike the other development technologies listed here, node.js is also its own web server software, thus eliminating the need for Apache, IIS, or some other web server software.
- **Perl.**
  - Until the development and popularization of ASP, PHP, and JSP, Perl was the language typically used for early server-side web development.
  - As a language, it excels in the manipulation of text.
  - It was commonly used in conjunction with the Common Gateway Interface (CGI), an early standard API for communication between applications and web server software.
- **PHP.**
  - Like ASP, PHP is a dynamically typed language that can be embedded directly within the HTML, though it now supports most common object oriented features, such as classes and inheritance.
  - By default, PHP pages are compiled into an intermediary representation called opcodes.

- Originally, PHP stood for personal home pages, but now it stands for **Hypertext Processor.**

➢ **Python.**

- This terse, object-oriented programming language has many uses, including being used to create web applications.
- It is also used in a variety of web development frameworks such as Django and Pyramid.

➢ **Ruby on Rails.**

- This is a web development framework that uses the Ruby programming language. Like ASP.NET and JSP, Ruby on Rails emphasizes the use of common software development approaches, in particular the MVC design pattern.
- It integrates features such as templates and engines that aim to reduce the amount of development work required in the creation of a new site.

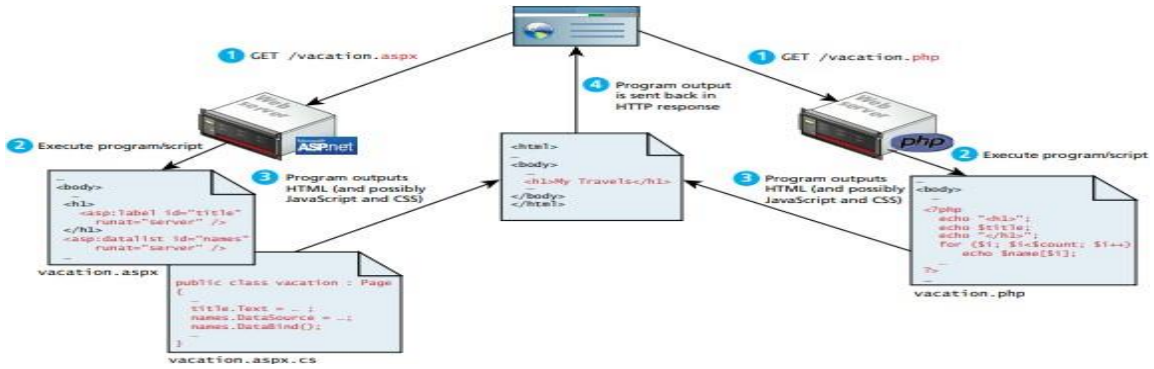


Figure 2.2: Web development technologies

A Web Server's Responsibilities

➢ A web server has many responsibilities beyond responding to requests for HTML files.

- These include handling HTTP connections.
- Responding to requests for static and dynamic resources.
- Managing permissions and access for certain resources.
- Encrypting and compressing data, managing multiple domains and URLs.
- Managing database connections.
- Cookies and state
- Uploading and managing files.

Apache and Linux

- Apache web server is the intermediary that interprets HTTP requests that arrive through a network port and decides how to handle the request, which often requires working in conjunction with PHP; both Apache and PHP make use of configuration files that determine exactly how requests are handled.
- Apache runs as a daemon on the server. A daemon is an executing instance of a program that runs in the background, waiting for a specific event that will activate it.
- As a background process, the Apache daemon waits for incoming HTTP requests. When a request arrives, Apache then uses modules to determine how to respond to the request.

Apache and PHP

- PHP is usually installed as an Apache module and PHP module mod_php5 is sometimes referred to as the SAPI (Server Application Programming Interface) layer since it handles the interaction between the PHP environment and the web server environment.

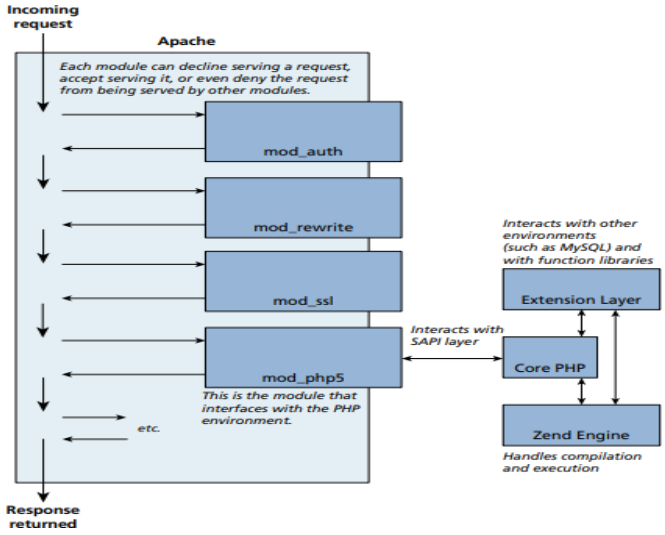


Figure 2.3: Apache modules and PHP

- Apache runs in two possible modes: multi-process (also called preforked) or multi-threaded (also called worker).

Multi-process:

- The default installation of Apache runs using the multi-process mode.
- That is, each request is handled by a separate process of Apache; the term fork refers to the operating system creating a copy of an already running process.
- A key advantage of multi-processing mode is that each process is insulated from other processes; that is, problems in one process can't affect other processes.

Multi-threaded:

- A smaller number of Apache processes are forked.
- Each of the processes runs multiple threads.
- A thread is like a lightweight process that is contained within an operating system process.
- A thread uses less memory than a process, and typically threads share memory and code.
- When using this mode, all modules running within Apache have to be thread safe.

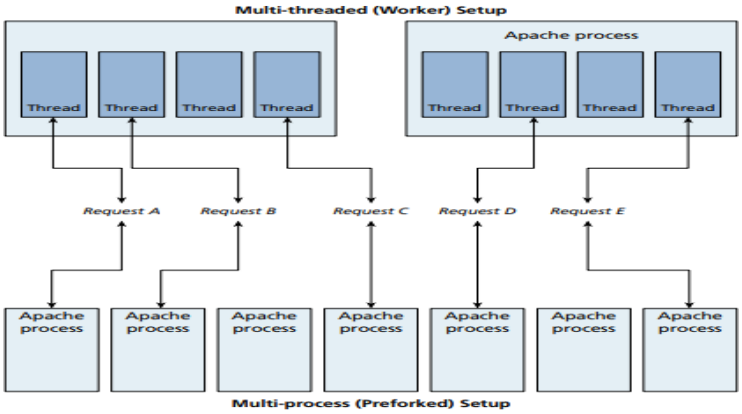


Figure 2.4: Multi-threaded versus multi-process

PHP Internals

1. PHP core.
   - The Core module defines the main features of the PHP environment, including essential functions for variable handling, arrays, strings, classes, math, and other core features.
2. Extension layer.
   - This module defines functions for interacting with services outside of PHP. This includes libraries for MySQL (and other databases), FTP, SOAP web services, and XML processing, among others.
3. Zend Engine.
   - This module handles the reading in of a requested PHP file, compiling it, and executing it.
   - Figure 2.4 illustrates how the Zend Engine operates behind the scenes when a PHP page is requested.
   - The Zend Engine is a virtual machine (VM) analogous to the Java Virtual Machine.

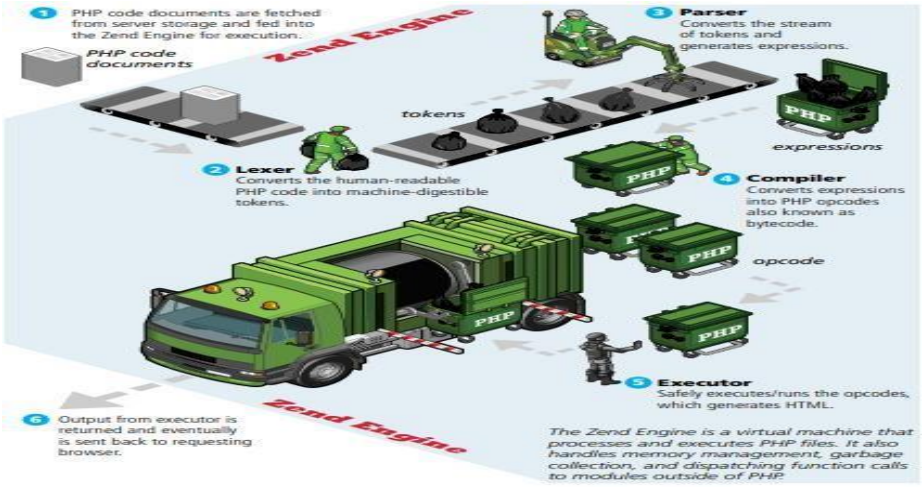


Figure 2.5: Zend Engine

Installing Apache, PHP, and MySQL for Local Development

- For Windows installation package (available at http://www.apachefriends.org/en/ xampp-windows.html).
- Once the XAMPP package is installed in Windows, we can then run the XAMPP control panel.
- We may need to click the appropriate Start buttons to launch Apache (and later MySQL).
- Once Apache has started, any subsequent PHP requests in our browser will need to use the localhost domain.
- Now we are ready to start creating our own PHP pages. If we used the default XAMPP installation location, our PHP files will have to be saved somewhere within the C:\xampp\htdocs folder.

Quick Tour of PHP

- It is a dynamically typed language.
- Unlike JavaScript it uses classes and functions in a way consistent with other object-oriented languages such as C++, C#, and Java, though with some minor exceptions.
- The syntax for loops, conditionals, and assignment is identical to JavaScript, only differs in functions, classes, and in how we define variables.

PHP Tags

- PHP code can be embedded directly within an HTML file.
- However, instead of having an **.html** extension, a PHP file will usually have the extension **.php**.

- PHP programming code must be contained within an opening<? Php tag and a matching closing?> tag in order to differentiate it from the HTML.
- The programming code within the tags is interpreted and executed, while any code outside the tags is echoed directly out to the client.

```
<?php
$user = "Randy";
?>
<!DOCTYPE html>
<html>
<body>
<h1>Welcome <?php echo $user; ?></h1>
<p>
The server time is
<?php
echo "<strong>";
echo date("H:i:s");
echo "</strong>";
?>
</p>
</body>
</html>
```

Listing 2.1: PHP tags

```
<!DOCTYPE html>
<html>
<body>
<h1>Welcome Randy</h1>
<p>
The server time is <strong>02:59:09</strong>
</p>
</body>
</html>
```

Listing 2.2: Listing 2.1 in browser

- We can separate the PHP code from HTML code as we have done in CSS and JS to increase the maintaince and accessibility by making use of include statements, which insert the content of the specified file into the current file.

Figure 2.6: Two approaches to PHP coding

PHP Comments

The types of comment styles in PHP are:

- **Single-line comments(#)**
  - Lines that begin with a # are comment lines and will not be executed.
- **Multiline (block) comments(/* */)**
  - These comments begin with a /* and encompass everything that is encountered until a closing */ tag is found.
  - These tags cannot be nested.
  - A comment block above a function or at the start of a file is a good place to write, in normal language, what this function does.
  - By using the /** tag to open the comment instead of the standard /*, we are identifying blocks of comment that can later be parsed for inclusion in generated documents.
- **End-of-line comments(//)**
  - Sometimes a variable needs a little blurb to tell the developer what it's for, or a complex portion of code needs a few comments to help the programmer understand the logic.
  - Whenever // is encountered in code, everything up to the end of the line is considered a comment. These comments are sometimes preferable to the block comments because they do not interfere with one another, but are unable to span multiple lines of code.

```php
<?php

# single-line comment

/*
This is a multiline comment.
They are a good way to document functions or complicated blocks of code
*/

$artist = readDatabase(); // end-of-line comment

?>
```

Listing 2.3: PHP comments

Variables, Data Types, and Constants

- Variables in PHP are dynamically typed, which means that we as a programmer do not have to declare the data type of a variable.
- Variables are also loosely typed in that a variable can be assigned different data types over time.
- To declare a variable we must preface the variable name with the dollar ($) symbol.

- Whenever we use that variable, we must also include the $ symbol with it.
- We can assign a value to a variable as in JavaScript's right-to-left assignment, so creating a variable named count and assigning it the value of 42 would be done with:

```
$count = 42;
```

- We should note that in PHP the name of a variable is case-sensitive, so **$count** and **$Count** are references to two different variables.
- In PHP, variable names can also contain the underscore character, which is useful for readability reasons.
- While PHP is loosely typed, it still does have data types, which describe the type of content that a variable can contain.
- Table 2.1 lists the main data types within PHP. We do not declare a data type. Instead the PHP engine determines the data type when the variable is assigned a value.
- String literals in PHP can be defined using either the single quote or the double quote character.
- If a literal is defined using double quotes, then you can also specify escape sequences using the backslash.

Data Type	Description
Boolean	A logical true or false value
Integer	Whole numbers
Float	Decimal numbers
String	Letters
Array	A collection of data of any type (covered in the next chapter)
Object	Instances of classes

Table 2.1: PHP Data Types

Sequence	Description
\n	Line feed
\t	Horizontal tab
\\	Backslash
\$	Dollar sign
\"	Double quote

Table 2.2: String Escape Sequences

Constant

- A constant is somewhat similar to a variable, except a constant's value never changes . . . in other words it stays constant.
- A constant can be defined anywhere but is typically defined near the top of a PHP file via the define() function, as shown in Listing 2.4.
- The define() function generally takes two parameters: the name of the constant and its value. Notice that once it is defined, it can be referenced without using the $ symbol.

```php
<?php

# uppercase for constants is a programming convention
define("DATABASE_LOCAL", "localhost");
define("DATABASE_NAME", "ArtStore");
define("DATABASE_USER", "Fred");
define("DATABASE_PASSWD", "F5^7%ad");
...
# notice that no $ prefaces constant names
$db = new mysqli(DATABASE_LOCAL, DATABASE_NAME, DATABASE_USER,
  DATABASE_NAME);

?>
```

Listing 2.4: PHP constants

Writing to Output

- Remember that PHP pages are programs that output HTML.
- To output something that will be seen by the browser, we can use the echo() function.

- There is also an equivalent shortcut version that does not require the parentheses.

```php
echo "hello";
```

- Strings can easily be appended together using the concatenate operator, which is the period (.) symbol.

```php
$username = "Ricardo";
echo "Hello". $username;
```

```php
<?php
$firstName = "Pablo";
$lastName = "Picasso";

/*
   Example one:
   These two lines are equivalent. Notice that you can reference PHP
   variables within a string literal defined with double quotes.

   The resulting output for both lines is:

     <em>Pablo Picasso</em>

*/
echo "<em>" . $firstName . " ". $lastName. "</em>";
echo "<em> $firstName $lastName </em>";

/*
   Example two:
   These two lines are also equivalent. Notice that you can use
   either the single quote symbol or double quote symbol for string
   literals.
*/
echo "<h1>";
echo '<h1>';

/*
   Example three:
   These two lines are also equivalent. In the second example, the
   escape character (the backslash) is used to embed a double quote
   within a string literal defined within double quotes.
*/
echo '<img src="23.jpg" >';
echo "<img src=\"23.jpg\" >";

?>
```

Listing 2.5: PHP quote usage and concatenation approaches

- The first example in above listing illustrates the fact that variable references can appear within string literals (but only if the literal is defined using double quotes).
- As such, it is important to take some time to experiment and evaluate some sample concatenation statements as shown in Listing 2.6.
- The output for the statements in Listing 2.6 is provided in figure 2.7.

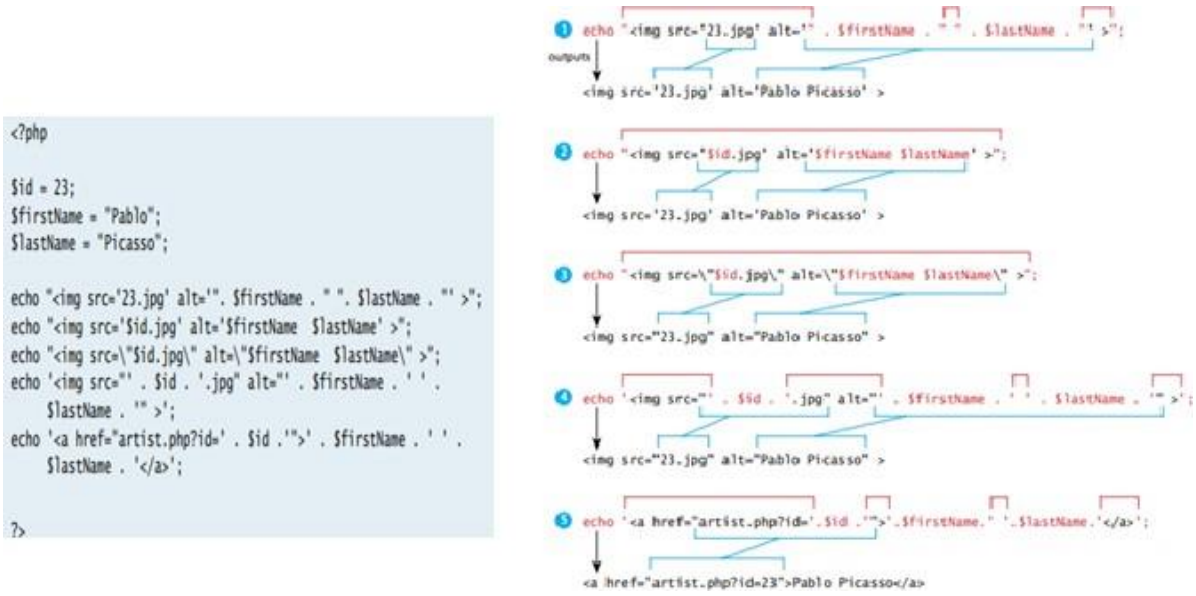


Figure 2.7: More Complicated concatenation examples Explained

Printf

- As the examples discussed above illustrate, while echo is quite simple, more complex output can get confusing.
- As an alternative, We can use the printf() function.
- The function takes at least one parameter, which is a string, and that string optionally references parameters, which are then integrated into the first string by placeholder substitution.
- The printf() function also allows a developer to apply special formatting, for instance, specific date/time formats or number of decimal places.
- Figure 2.8 illustrates the relationship between the first parameter string, its placeholders and subsequent parameters, precision, and output.

Figure 2.8: Illustration of components in a printf statement and output

- Each placeholder requires the percent (%) symbol in the first parameter string followed by a type specifier.
- Common type specifiers are b for binary, d for signed integer, f for float, o for octal, and x for hexadecimal.
- Precision is achieved in the string with a period (.) followed by a number specifying how many digits should be displayed for floating-point numbers.

- When programming, we may prefer to use printf() for more complicated formatted output, and use echo for simpler output.

Program Control

- Just as with most other programming languages there are a number of conditional and iteration constructs in PHP.
- There are if and switch, and while, do while, and for loops familiar to most languages as well as the for each loop.

if...else

- The syntax for conditionals in PHP is almost identical to that of JavaScript.
- In this syntax the condition to test is contained within() brackets with the body contained in{}blocks.
- Optional else if statements can follow, with an else ending the branch as in Listing 2.6.

```
// if statement with condition
if ( $hourOfDay > 6 && $hourOfDay < 12 ) {
   $greeting = "Good Morning";
}
else if ($hourOfDay == 12) {   // optional else if
   $greeting = "Good Noon Time";
}
else {                         // optional else branch
   $greeting = "Good Afternoon or Evening";
}
```

Listing 2.6: if . . . else

```
<?php if ($userStatus == "loggedin") {  ?>
   <a href="account.php">Account</a>
   <a href="logout.php">Logout</a>
<?php }  else { ?>
   <a href="login.php">Login</a>
   <a href="register.php">Register</a>
<?php } ?>


<?php
   // equivalent to the above conditional
   if ($userStatus == "loggedin") {
      echo '<a href="account.php">Account</a> ';
      echo '<a href="logout.php">Logout</a>';
   }
   else {
      echo '<a href="login.php">Login</a> ';
      echo '<a href="register.php">Register</a>';
   }
?>
```

Listing 2.7: Combining PHP and HTML in the script

- It is also possible to place the body of an if or an else outside of PHP as in Listing 2.7.

switch...case

The switch statement is similar to a series of if...else statements as in Listing 2.8.

```php
switch ($artType) {
    case "PT":
        $output = "Painting";
        break;
    case "SC":
        $output = "Sculpture";
        break;
    default:
        $output = "Other";
}

// equivalent
if ($artType == "PT")
    $output = "Painting";
else if ($artType == "SC")
    $output = "Sculpture";
else
    $output = "Other";
```

Listing 2.8: Conditional statement using switch

```php
$count = 0;
while ($count < 10)
{
    echo $count;
    $count++;
}

$count = 0;
do
{
    echo $count;
    $count++;
} while ($count < 10);
```

Listing 2.9: while loops

While and do...while

- The while loop and the do...while loop are quite similar. Both will execute nested statements repeatedly as long as the while expression evaluates to true.
    - In the while loop, the condition is tested at the beginning of the loop;
    - in the do... while loop the condition is tested at the end of each iteration of the loop.
    - Listing 2.9 provides examples of each type of loop.

For

- The for loop in PHP has the same syntax as the for loop in JavaScript that we examined and the for loop contains the same loop initialization, condition, and post-loop operations as in JavaScript.
- There is another type of for loop: the foreach loop. This loop is especially useful for iterating through arrays.

```php
for ($count=0; $count < 10; $count++)
{
   echo $count;
}
```

Listing 2.10: for loops

```php
<?php if ($userStatus == "loggedin") :  ?>
   <a href="account.php">Account</a>
   <a href="logout.php">Logout</a>
<?php else : ?>
   <a href="login.php">Login</a>
   <a href="register.php">Register</a>
<?php endif; ?>
```

Listing 2.11: Alternate syntax for control structures

Alternate Syntax for Control Structures

- PHP has an alternative syntax for most of its control structure PHP has an alternative syntax for most of its control structures (namely, the if, while, for, foreach, and switch statements).
- In this alternate syntax (shown in Listing 2.11), the colon (:) replaces the opening curly bracket, while the closing brace is replaced with endif;, endwhile;, endfor;, endforeach;, or endswitch; as in Listing 2.11.

Include Files

- PHP does have one important facility that is generally unlike other nonweb programming languages, namely the ability to include or insert content from one file into another.

- Include files provide a mechanism for reusing both markup and PHP code, as shown in Figure 2.9

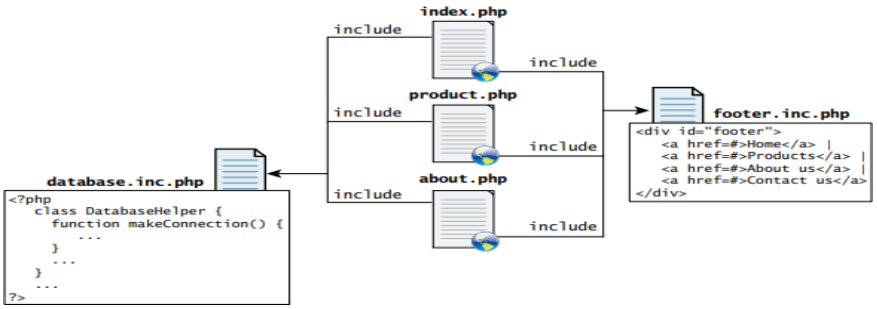


Figure 2.9: Include Files

- PHP, include files are essentially the only way to achieve code and markup reuse.
- PHP provides four different statements for including files, as shown below.

```
include "somefile.php";
include_once "somefile.php";

require "somefile.php";
require_once "somefile.php";
```

- The difference between include and require lies in what happens when the specified file cannot be included.

Include

- With include, a warning is displayed and then execution continues.
- The include_once statements works as include but if the requested file has already been included once, then it will not be included again.

Require

- With require, an error is displayed and execution stops.
- The require_once statements works as include but if the requested file has already been included once, then it will not be included again.

Scope within Include Files

- Include files appear to provide a type of encapsulation, but it is important to realize that they are the equivalent of copying and pasting, though in this case it is performed by the server.
- Variables defined within an include file will have the scope of the line on which the include occurs.
- Any variables available at that line in the calling file will be available within the called file.
- If the include occurs inside a function, then all of the code contained in the called file will behave as though it had been defined inside that function.
- Thus, for true encapsulation, you will have to use functions and classes.

Functions

- Just as with any language, writing code in the main function (which in PHP is equivalent to coding in the markup between <? Php and?>tags) is not a good habit to get into.
- Having all our code in the main body of a script makes it hard to reuse, maintain, and understand.
- As an alternative, PHP allows us to define functions. Just like with JavaScript, a **function** in PHP contains a small bit of code that accomplishes one thing.
- These functions can be made to behave differently based on the values of their parameters.
- Functions can exist all on their own, and can then be called from anywhere that needs to make use of them, so long as they are in scope.
- Later we will write functions inside of classes, which we will call methods.
- In PHP there are two types of function: user-defined functions and built-in functions.
  - A **user-defined function** is one that you the programmer define.
  - A **built-in function** is one of the functions that come with the PHP environment (or with one of its extensions)
- One of the real strengths of PHP is its rich library of built-in functions that you can use.

Function Syntax

- To create a new function we must think of a name for it, and functions can return values to the caller, or not return a value.
- They can be setup to take or not take parameters. To illustrate function syntax, let us examine a function called getNiceTime(), which will return a formatted string containing the current server time.
- We will notice that the definition requires the use of the function key word followed by the function's name, round( ) brackets for parameters, and then the body of the function inside curly{} brackets.
- Listing 8.13 returns a value and Listing 8.14 illustrates a function definition that doesn't return a value but just performs a task.

```php
/**
* This function returns a nicely formatted string using the current
* system time.
*/
function getNiceTime() {
   return date("H:i:s");
}
```

Listing 2.12: function to return value

```php
/**
* This function outputs the footer menu
*/
function outputFooterMenu() {
   echo '<div id="footer">';
   echo '<a href=#>Home</a> | <a href=#>Products</a> | ';
   echo '<a href=#>About us</a> | <a href=#>Contact us</a>';
   echo '</div>';
}
```

Listing 2.13: function without a return value

Calling a Function

- We have defined a function, we are able to use it whenever you want to.
- To call a function you must use its name with the () brackets.
- Since getNiceTime() returns a string, you can assign that return value to a variable, or echo that return value directly, as shown below

```
$output = getNiceTime();
echo getNiceTime();
```

- If the function doesn't return a value, you can just call the function.

```
outputFooterMenu();
```

Parameters

- It is more common to define functions with parameters, since functions are more powerful and reusable when their output depends on the input they get.
- Parameters are the mechanism by which values are passed into functions, and there are some complexities that allow us to have multiple parameters, default values, and to pass objects by reference instead of value.
- To define a function with parameters, you must decide how many parameters you want to pass in, and in what order they will be passed. Each parameter must be named.
- As shown in Listing 2.14. Notice that parameters, being a type of variable, must be prefaced with a $ symbol like any other PHP variable.

```
/**
* This function returns a nicely formatted string using the current
* system time. The showSeconds parameter controls whether or not to
* include the seconds in the returned string.
*/
function getNiceTime($showSeconds) {
  if ($showSeconds==true)
    return date("H:i:s");
  else
    return date("H:i");
}
```

Listing 2.14: A function to return the current time as a string with an integer parameter

- Thus to call our function, you can now do it in two ways:

```
echo getNiceTime(1);    // this will print seconds
echo getNiceTime(0);    //  will not print seconds
```

Parameter Default Values

- If we could not simply combine the two overloaded functions together into one so that if we call it with no parameter, it uses a default value.
- In PHP you can set parameter default values for any parameter in a function. However, once you start having default values, all subsequent parameters must also have defaults.
- Listing 2.15 illustrate the default values.

```php
/**
* This function returns a nicely formatted string using the current
* system time. The showSeconds parameter controls whether or not
* to show the seconds.
*/
function getNiceTime($showSeconds=1){
   if ($showSeconds==true)
      return date("H:i:s");
   else
      return date("H:i");
}
```

Listing 2.15: A function to return the current time with a parameter that includes a default

- If you do include a value in your function call, the default will be overridden by whatever that value was.
- Either way we can have a single function that can be called with or without values passed.

Passing Parameters by Reference

- By default, arguments passed to functions are **passed by value** in PHP. This means that PHP passes a copy of the variable so if the parameter is modified within the function, it does not change the original.
- Listing 2.16 illustrates a simple example of passing by value.

```php
function changeParameter($arg) {
   $arg += 300;
   echo "<br/>arg=" . $arg;
}

$initial = 15;
echo "<br/>initial=" . $initial;    // output: initial=15
changeParameter($initial);          // output: arg=315
echo "<br/>initial=" . $initial;    // output: initial=15
```

Listing 2.16: Passing a parameter by value

```php
function changeParameter(&$arg) {
   $arg += 300;
   echo "<br/>arg=". $arg;
}

$initial = 15;
echo "<br/>initial=" . $initial;    // output: initial=15
changeParameter($initial);          // output: arg=315
echo "<br/>initial=" . $initial;    // output: initial=315
```

Listing 2.17: Passing a parameter by reference

- PHP also allows arguments to functions to be **passed by reference**, which will allow a function to change the contents of a passed variable.
- A parameter passed by reference points the local variable to the same place as the original, so if the function changes it, the original variable is changed as well.
- The mechanism in PHP to specify that a parameter is passed by reference is to add an ampersand (&) symbol next to the parameter name in the function declaration.
- Listing 2.17 illustrates an example of passing by reference.
- Figure 2.10 illustrates visually the memory differences between pass-by value and pass-by reference.
- By and large, we will likely find that most of the time we will use pass-by value in the majority of our functions.

Variable Scope within Functions

- It will come as no surprise that all variables defined within a function (such as parameter variables) have function scope, meaning that they are only accessible within the function.
- It might be surprising though to learn that any variables created outside of the function in the main script are unavailable within a function.

```
$count= 56;

function testScope() {
   echo $count;     // outputs 0 or generates run-time warning/error
}
testScope();
echo $count;       // outputs 56
```

- While variables defined in the main script are said to have global scope, unlike in other programming languages, a global variable is not, by default, available within functions.
- For instance, most web applications will have important data values such as connections, application constants, and logging/debugging switches that need to be available throughout the application, and passing them to every function that might need them is often impractical.
- But PHP does allow variables with global scope to be accessed within a function using the global keyword, as shown in Listing 2.18.

```
$count= 56;

function testScope() {
   global $count;
   echo $count;     // outputs 56
}

testScope();
echo $count;       // outputs 56
```

Listing 2.18: Using the global keyword

- From a programming design standpoint, the use of global variables should be minimized, and only used for vital application objects that are truly global.

MODULE 4

Chapter 1: PHP Arrays and Superglobals

1. Arrays
2. $_GET and $_POST Superglobal Arrays
3. $_SERVER Array
4. $_Files Array
5. Reading/Writing Files

Arrays

- An array is a data structure that allows the programmer to collect a number of related elements together in a single variable.
- Unlike most other programming languages, in PHP an array is actually an **ordered map**, which associates each value in the array with a key.
- This flexibility allows us to use arrays in PHP in a manner similar to other languages' arrays, but we can also use them like other languages' collection classes.

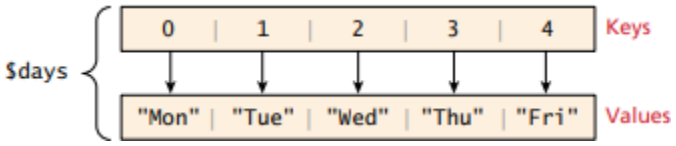


Figure 1.1: Visualization of a key-value array

- **Array keys**
  - In most programming languages array keys are limited to integers, i.e., start at 0,and go up by 1.
  - In PHP, keys *must* be either integers or strings and need not be sequential.
  - This means we cannot use an array or object as a key (doing so will generate an error).
  - One should be especially careful about mixing the types of the keys for an array since PHP performs cast operations on the keys that are not integers or strings.
- **Array values**
  - Unlike keys, are not restricted to integers and strings.

- They can be any object, type, or primitive supported in PHP.
- We can even have objects of our own types, so long as the keys in the array are integers and strings.

Defining and Accessing an Array

- Let us begin by considering the simplest array, which associates each value inside of it with an integer index (starting at 0).
- The following declares an empty array named days:

```
$days = array();
```

- To define the contents of an array as strings for the days of the week as shown in Figure 1.1, we declare it with a comma-delimited list of values inside the ( )braces using either of two following syntaxes:

```
$days = array("Mon","Tue","Wed","Thu","Fri");
$days = ["Mon","Tue","Wed","Thu","Fri"];     // alternate syntax
```

- In the above example, because no keys are explicitly defined for the array, the default key values are 0, 1, 2, . . . , n.
- Notice that we do not have to provide a size for the array: arrays are dynamically sized as elements are added to them.
- Elements within a PHP array are accessed in a manner similar to other programming languages, that is, using the familiar square bracket notation.
- The code example below echoes the value of our $days array for the key=1, which results in output of Tue.

```
echo "Value at index 1 is ". $days[1];    // index starts at zero
```

- We could also define the array elements individually using the same square bracket notation:

```
$days = array();
$days[0] = "Mon";
$days[1] = "Tue";
$days[2] = "Wed";

// also alternate approach
$daysB = array();
$daysB[] = "Mon";
$daysB[] = "Tue";
$daysB[] = "Wed";
```

- In PHP, we are also able to explicitly define the keys in addition to the values.
- This allows us to use keys other than the classic 0, 1, 2, . . , n to define the indexes of an array.

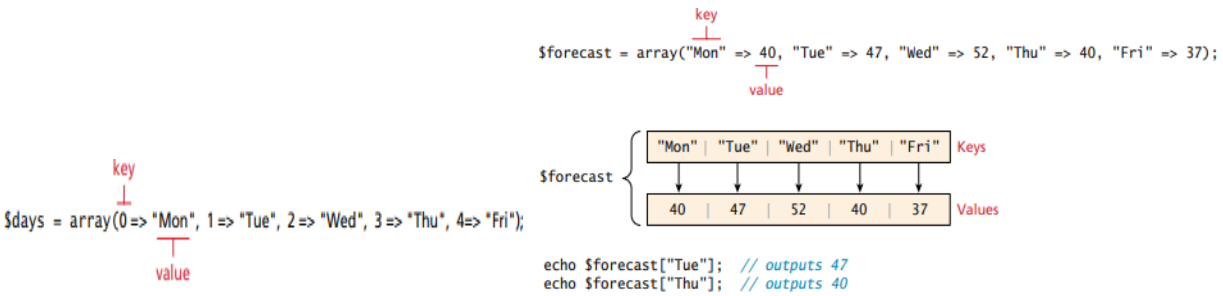


Figure 1.2: Assigning keys to values (left) and Array with strings as keys and integers as values (left)

- Explicit control of the keys and values opens the door to keys that do not start at 0, are not sequential, and that are not even integers (but rather strings).
- This is why we can also consider an array to be a dictionary or hash map.
- These types of arrays in PHP are generally referred to as **associative arrays**.
- Keys must be either integer or string values, but the values can be any type of PHP data type, including other arrays. e
- To access an element in an associative array, we simply use the key value rather than an index. `echo $forecast["Wed"]; // this will output 52`

Multidimensional Arrays

- PHP also supports multidimensional arrays.
- The values for an array can be any PHP object, which includes other arrays.

```
$month = array
  (
  array("Mon","Tue","Wed","Thu","Fri"),
  array("Mon","Tue","Wed","Thu","Fri"),
  array("Mon","Tue","Wed","Thu","Fri"),
  array("Mon","Tue","Wed","Thu","Fri")
  );

echo $month[0][3];   // outputs Thu
```

```
$cart = array();
$cart[] = array("id" => 37, "title" => "Burial at Ornans",
                "quantity" => 1);
$cart[] = array("id" => 345, "title" => "The Death of Marat",
                "quantity" => 1);
$cart[] = array("id" => 63, "title" => "Starry Night", "quantity" => 1);

echo $cart[2]["title"];  // outputs Starry Night
```

Figure 1.3: Illustrates the structure of these two multidimensional arrays.

Iterating through an Array

- One of the most common programming tasks that we will perform with an array is to iterate through its contents.

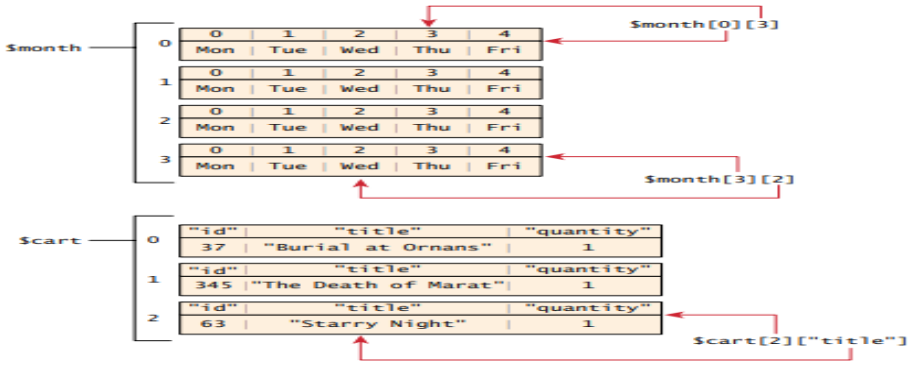


Figure 1.4: Visualizing multidimensional arrays

- Listing 1.1 illustrates how to iterate and output the content of the $days array using the built-in function count() along with examples using while, do while, and for loops.

```php
// while loop
$i=0;
while ($i < count($days)) {
    echo $days[$i] . "<br>";
    $i++;
}

// do while loop
$i=0;
do {
    echo $days[$i] . "<br>";
    $i++;
} while ($i < count($days));

// for loop
for ($i=0; $i<count($days); $i++) {
    echo $days[$i] . "<br>";
}
```

Listing 1.1: Iterating an array using while, do while, and for loops

```php
// foreach: iterating through the values
foreach ($forecast as $value) {
   echo $value . "<br>";
}

// foreach: iterating through the values AND the keys
foreach ($forecast as $key => $value) {
   echo "day" . $key . "=" . $value;
}
```

Listing 1.2: Iterating through an associative array using a foreach loop

- The challenge of using the classic loop structures is that when we have nonsequential integer keys (i.e., an associative array), we can't write a simple loop that uses the $i++ construct.
- To overcome this challenge we need to use iterator to move through an associative array as shown in Listing 1.2.

Adding and Deleting Elements

- In PHP, arrays are dynamic, i.e., they can grow or shrink in size.
- An element can be added to an array simply by using a key/index that hasn't been used, as shown below:

```php
$days[5] = "Sat";
```

- Since there is no current value for key 5, the array grows by one, with the new key/value pair added to the end of our array.
- If the key had a value already, the same style of assignment replaces the value at that key.

- As an alternative to specifying the index, a new element can be added to the end of any array using the following technique:

```
$days[ ] = "Sun";
```

- The advantage to this approach is that we don't have to worry about skipping an index key. PHP is more than happy to let us "skip" an index, as shown in the following example.

```
$days = array("Mon","Tue","Wed","Thu","Fri");
$days[7] = "Sat";
print_r($days);
```

- Output for the above snippet is:

```
Array ([0] => Mon [1] => Tue [2] => Wed [3] => Thu [4] => Fri [7] => Sat)
```

  - That is, there is now a "gap" in our array that will cause problems if we try iterating through it using the techniques.
  - If we try referencing $days[6], for instance, it will return a **NULL** value, which is a special PHP value that represents a variable with no value.

- We can also create "gaps" by explicitly deleting array elements using the unset() function as shown in Listing 1.3.

```
$days = array("Mon","Tue","Wed","Thu","Fri");

unset($days[2]);
unset($days[3]);

print_r($days); // outputs: Array ( [0] => Mon [1] => Tue [4] => Fri )

$days = array_values($days);
print_r($days); // outputs: Array ( [0] => Mon [1] => Tue [2] => Fri )
```

Listing 1.3: Deleting elements

```
$oddKeys = array (1 => "hello", 3 => "world", 5 => "!");
if (isset($oddKeys[0])) {
   // The code below will never be reached since $oddKeys[0] is not set!
   echo "there is something set for key 0";
}
if (isset($oddKeys[1])) {
   // This code will run since a key/value pair was defined for key 1
   echo "there is something set for key 1, namely ". $oddKeys[1];
}
```

Listing 1.4: nonsequential keys and usage of isset( )

- **Checking If a Value Exists:**
  - Since array keys need not be sequential, and need not be integers, we may run into a scenario where we want to check if a value has been set for a particular key.
  - As with undefined null variables, values for keys that do not exist are also undefined.
  - To check if a value exists for a key, we can therefore use the isset() function, which returns true if a value has been set, and false otherwise.
  - Listing 1.4 defines an array with noninteger indexes, and shows the result of asking isset() on several indexes.

Array Sorting

- There are many built-in sort functions, which sort by key or by value. To sort the $days array by its values we would simply use:

```
sort($days);
```

  - As the values are all strings, the resulting array would be:

```
Array ([0] => Fri [1] => Mon [2] => Sat [3] => Sun [4] => Thu
       [5] => Tue [6] => Wed)
```

- However, the above sort loses the association between the values and the keys!
- A better sort, one that would have kept keys and values associated together, is:

```
asort($days);
```

  - The resulting array in this case is:

```
Array ([4] => Fri [0] => Mon [5] => Sat [6] => Sun [3] => Thu
       [1] => Tue [2] => Wed)
```

More Array Operations

In addition to the powerful sort functions, there are other convenient functions you can use on arrays.

- **array_keys($someArray**): This method returns an indexed array with the values being the keys of $someArray.
  - For example, print_r(array_values($days)) outputs

```
Array ( [0] => 0 [1] => 1 [2] => 2 [3] => 3 [4] => 4 )
```

- **array_values($someArray):** This function returns an indexed array with the values being the values of $someArray.
  - For example, print_r(array_values($days)) outputs

```
Array ( [0] => Mon [1] => Tue [2] => Wed [3] => Thu [4] => Fri )
```

- **array_rand($someArray, $num=1):** This function returns as many random keys as are requested. If we only want one, the key itself is returned; otherwise, an array of keys is returned.
  - For example, print_r(array_rand($days,2)) might output: `Array (3, 0)`
- **array_reverse($someArray):** This method returns $someArray in reverse order. The passed $someArray is left untouched.

- For example, print_r(array_reverse($days)) outputs:

```
Array ( [0] => Fri [1] => Thu [2] => Wed [3] => Tue [4] => Mon )
```

- **array_walk($someArray, $callback, $optionalParam):** It allows us to call a method ($callback), for each value in $someArray. The $callback function typically takes two parameters, the value first, and the key second.
  - An example that simply prints the value of each element in the array is shown below.

```
$someA = array("hello", "world");
array_walk($someA, "doPrint");
function doPrint($value,$key){
  echo $key . ": " . $value;
}
```

- **in_array($needle, $haystack):** This method lets us to search array $haystack for a value ($needle). It returns true if it is found, and false otherwise.
- **shuffle($someArray):** This method shuffles $someArray. Any existing keys are removed and $someArray is now an indexed array if it wasn't already.

Superglobal Arrays

- PHP uses special predefined associative arrays called superglobal variables that allow the programmer to easily access HTTP headers, query string parameters, and other commonly needed information.
- They are called superglobal because these arrays are always in scope and always exist, ready for the programmer to access or modify them without having to use the global keyword.

Name	Description
$GLOBALS	Array for storing data that needs superglobal scope
$_COOKIES	Array of cookie data passed to page via HTTP request
$_ENV	Array of server environment data
$_FILES	Array of file items uploaded to the server
$_GET	Array of query string data passed to the server via the URL
$_POST	Array of query string data passed to the server via the HTTP header
$_REQUEST	Array containing the contents of $_GET, $_POST, and $_COOKIES
$_SESSION	Array that contains session data
$_SERVER	Array containing information about the request and the server

Table 1.1: Superglobal variables

$_GET and $_POST Superglobal Arrays

- The $_GET and $_POST arrays are the most important superglobal variables in PHP since they allow the programmer to access data sent by the client in a query string.
- An HTML form (or an HTML link) allows a client to send data to the server.
- That data is formatted such that each value is associated with a name defined in the form.
- If the form was submitted using an HTTP GET request, then the resulting URL will contain the data in the query string.
- PHP will populate the superglobal $_GET array using the contents of this query string in the URL.

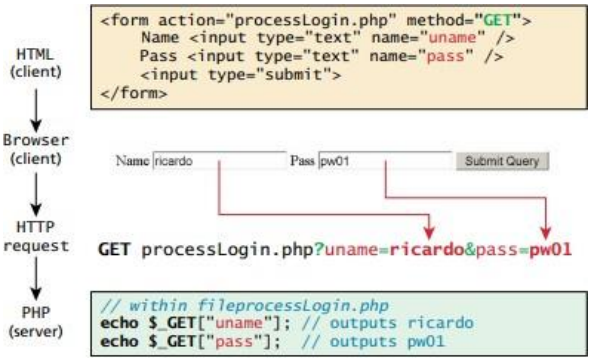


Figure1.5: HTTP request to $_GET array

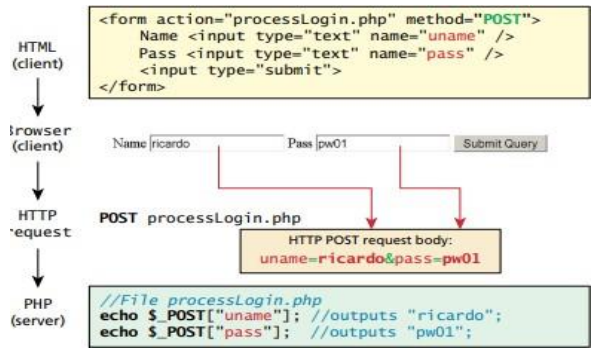


Figure1.6: HTTP request to $_POST array

- If the form was sent using HTTP POST, then the values would not be visible in the URL, but will be sent through HTTP POST request body.
- From the PHP programmer's perspective, almost nothing changes from a GET data post except that those values and keys are now stored in the $_POST array.
- This mechanism greatly simplifies accessing the data posted by the user, since we need not parse the query string or the POST request headers.

Determining If Any Data Sent

- PHP can use the same file to handle both the display of a form as well as the form input.
  - For **example**, a single file is often used to display a login form to the user, and that same file also handles the processing of the submitted form data, as shown in Figure 1.7.
  - In such cases we may want to know whether any form data was submitted at all using either POST or GET.

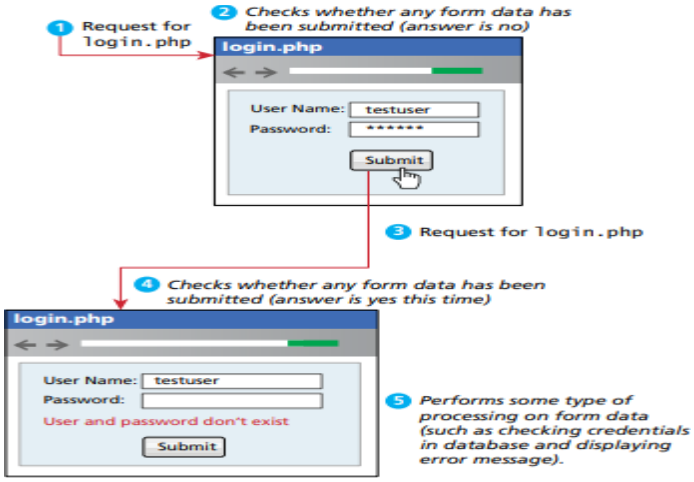


Figure 1.7: Form display and processing by the same PHP page

Accessing Form Array Data

- Sometimes in HTML forms you might have multiple values associated with a single name;
- Listing 1.5 provides another example. Notice that each checkbox has the same name value (name="day").

```
<form method="get">
   Please select days of the week you are free.<br />
   Monday <input type="checkbox" name="day" value="Monday" /> <br />
   Tuesday <input type="checkbox" name="day" value="Tuesday" /> <br />
   Wednesday <input type="checkbox" name="day" value="Wednesday" /> <br />
   Thursday <input type="checkbox" name="day" value="Thursday" /> <br />
   Friday <input type="checkbox" name="day" value="Friday" /> <br />
   <input type="submit" value="Submit">
</form>
```

Listing 1.5: HTML that enables multiple values for one name

```
<?php

echo "You submitted " . count($_GET['day']) . "values";
foreach ($_GET['day'] as $d) {
   echo $d . ", ";
}

?>
```

Listing 1.6: PHP code to display an array of checkbox variables

- If the user selects more than one day and submits the form, the $_GET['day'] value in the superglobal array will only contain the last value from the list that was selected.
- To overcome this limitation, you must change the HTML in the form. In particular, you will have to change the name attribute for each checkbox from day to day[].

```
Monday <input type="checkbox" name="day[]" value="Monday" />
Tuesday <input type="checkbox" name="day[]" value="Tuesday" />
...
```

- Listing 1.6 to echo the number of days selected and their values.

Using Query Strings in Hyperlinks

- WKT, form information packaged in a query string is transported to the server in one of two locations depending on whether the form method is GET or POST.
- It is also important to realize that making use of query strings is not limited to only data entry forms.
- It is possible to combine query strings with anchor tags . . Anchor tags (i.e., hyperlinks) also use the HTTP GET method
  - **EX:** Our database may have hundreds or thousands of books in it: surely it would be too much work to create a separate page for each book!
  - It would make a lot more sense to have a single Display Book page that receives as input a query string that specifies which book to display, as shown in Figure 1.8.

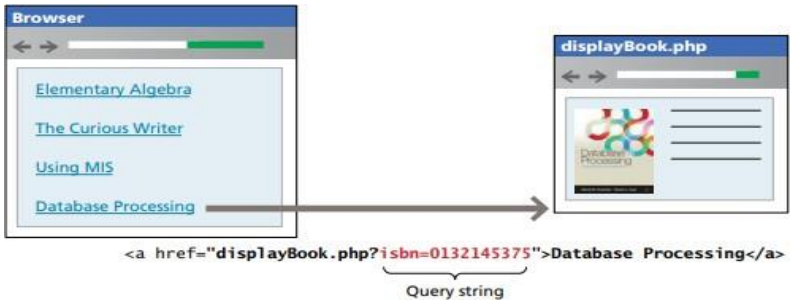


Figure 1.8: Sensible approach to displaying individual items using query strings

Sanitizing Query Strings

- The process of checking user input for incorrect or missing information is sometimes referred to as the process of sanitizing user inputs.

```
// This uses a database API . . . we will learn about it in Chapter 11
$pid = mysqli_real_escape_string($link, $_GET['id']);

if ( is_int($pid) ) {
     // Continue processing as normal
}
else {
     // Error detected. Possibly a malicious user
}
```

Listing 1.7: Simple sanitization of query string values

- Our program must be able to handle the following cases for every query string or form value.
  - If query string parameter doesn't exist.
  - If query string parameter doesn't contain a value.

- If query string parameter value isn't the correct type.
- If value is required for a database lookup, but provided value doesn't exist in the database table.

$_SERVER Array

- The $_SERVER associative array contains a variety of information as follows.
  - It contains some of the information contained within HTTP request headers sent by the client.
  - It also contains many configuration options for PHP itself as shown in figure 1.9

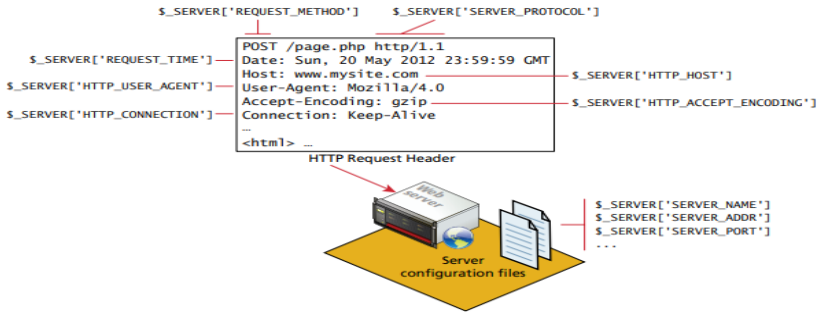


Figure 1.9: Relationship between request headers, the server, and the $_SERVER array

- To use the $_SERVER array, we simply refer to the relevant case-sensitive key name:

```
echo $_SERVER["SERVER_NAME"] . "<br/>";
echo $_SERVER["SERVER_SOFTWARE"] . "<br/>";
echo $_SERVER["REMOTE_ADDR"] . "<br/>";
```

- It is worth noting that because the entries in this array are created by the webserver, not every key listed in the PHP documentation will necessarily be available.
- A complete list of keys contained within this array is listed in the online PHP documentation, but we will cover some of the critical ones here.

Server Information Keys

- SERVER_NAME is a key in the $_SERVER array that contains the name of the site that was requested. If we are running multiple hosts on the same code base, this can be a useful piece of information.
- SERVER_ADDR is a complementary key telling us the IP of the server. Either of these keys can be used in a conditional to output extra HTML to identify a development server.

- DOCUMENT_ROOT tells us the file location from which we are currently running our script.
- Since we are often moving code from development to production, this key can be used to great effect to create scripts that do not rely on a particular location to run correctly.
- This key complements the SCRIPT_NAME key that identifies the actual script being executed.

Request Header Information Keys

- Recall that the web server responds to HTTP requests, and that each request contains a request header.
- These keys provide programmatic access to the data in the request header.
- The REQUEST_METHOD key returns the request method that was used to access the page: that is, GET, HEAD, POST, PUT.
- The REMOTE_ADDR key returns the IP address of the requestor, which can be a useful value to use in your web applications.
- In real-world sites these IP addresses are often stored to provide an audit trail of which IP made which requests, especially on sensitive matters like finance and personal information.

```php
<?php
echo $_SERVER['HTTP_USER_AGENT'];

$browser = get_browser($_SERVER['HTTP_USER_AGENT'], true);
print_r($browser);
?>
```

```php
$previousPage = $_SERVER['HTTP_REFERER'];
// Check to see if referer was our search page
if (strpos("search.php",$previousPage) != 0) {
   echo "<a href='search.php'>Back to search</a>";
}
// Rest of HTML output
```

Listing 1.8: Accessing the user-agent string in the HTTP headers

Listing 1.9: Using the HTTP_REFERER header to provide context-dependent output

- One of the most commonly used request headers is the user-agent header, which contains the operating system and browser that the client is using. This header value can be accessed using the key HTTP_USER_AGENT as shown in Listing 1.8.
- HTTP_REFERER is an especially useful header. Its value contains the address of the page that referred us to this one (if any) through a link. It is commonly used in analytics to determine which pages are linking to our site as shown in Listing 1.9.

$_FILES Array

- The $_FILES associative array contains items that have been uploaded to the current script.
- The <input type="file"> element is used to create the user interface for uploading a file from the client to the server. The user interface is only one part of the uploading process.

HTML Required for File Uploads

- To allow users to upload files, there are some specific things you must do:
- First, we must ensure that the HTML form uses the HTTP POST method, since transmitting a file through the URL is not possible.
- Second, we must add the enctype="multipart/form-data" attribute to the HTML form that is performing the upload so that the HTTP request can submit multiple pieces of data.
- Finally we must include an input type of file in our form. This will show up with a browse button beside it so the user can select a file from their computer to be uploaded.

```
<form enctype='multipart/form-data' method='post'>
   <input type='file' name='file1' id='file1' />
   <input type='submit' />
</form>
```

Listing 1.10: HTML for a form that allows an upload

Handling the File Upload in PHP

- The corresponding PHP file responsible for handling the upload will utilize the superglobal $_FILES array.
- This array will contain a key=value pair for each file uploaded in the post.
  - The key for each element will be the name attribute from the HTML form,
  - while the value will be an array containing information about the file as well as the file itself.
  - The keys in that array are the name, type, tmp_name, error, and size.

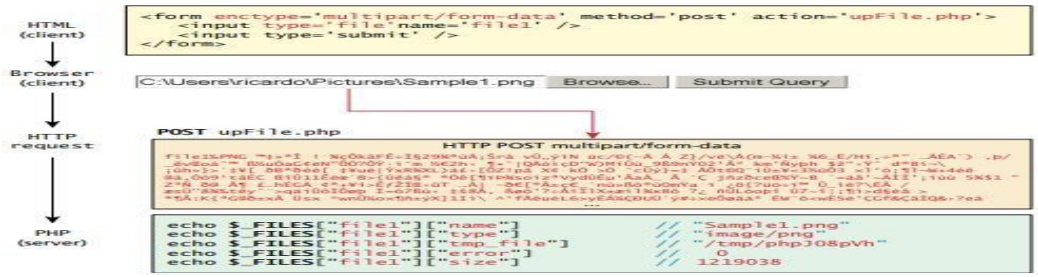


Figure 1.10: Data flow from HTML form through POST to PHP $_FILES array

- Figure 1.10 illustrates the process of uploading a file to the server and how the corresponding upload information is contained in the $_FILES array.
- The values for each of the keys, in general, are described below.
  - **Name** is a string containing the full file name used on the client machine, including any file extension. It does not include the file path on the client's machine.
  - **Type** defines the MIME type of the file. This value is provided by the client browser and is therefore not a reliable field.
  - **tmp_name** is the full path to the location on your server where the file is being temporarily stored. The file will cease to exist upon termination of the script, so it should be copied to another location if storage is required.
  - **error** is an integer that encodes many possible errors and is set to UPLOAD_ERR_OK (integer value 0) if the file was uploaded successfully.
  - **size** is an integer representing the size in bytes of the uploaded file.

Checking for Errors

- For every uploaded file, there is an error value associated with it in the $_FILES array.
- The error values are specified using constant values, which resolve to integers.
- The value for a successful upload is UPLOAD_ERR_OK, and should be looked for before proceeding any further.
- The full list of errors is provided in Table 1.2 and shows that there are many causes for bad file uploads.

Error Code	Integer	Meaning
UPLOAD_ERR_OK	0	Upload was successful.
UPLOAD_ERR_INI_SIZE	1	The uploaded file exceeds the upload_max_filesize directive in php.ini.
UPLOAD_ERR_FORM_SIZE	2	The uploaded file exceeds the max_file_size directive that was specified in the HTML form.
UPLOAD_ERR_PARTIAL	3	The file was only partially uploaded.
UPLOAD_ERR_NO_FILE	4	No file was uploaded. Not always an error, since the user may have simply not chosen a file for this field.
UPLOAD_ERR_NO_TMP_DIR	6	Missing the temporary folder.
UPLOAD_ERR_CANT_WRITE	7	Failed to write to disk.
UPLOAD_ERR_EXTENSION	8	A PHP extension stopped the upload.

Table 1.2: Error Codes in PHP for File Upload

```
foreach ($_FILES as $fileKey => $fileArray) {
  if ($fileArray["error"] != UPLOAD_ERR_OK) { // error
    echo "Error: " . $fileKey . " has error" . $fileArray["error"]
        . "<br>";
  }
  else {   // no error
    echo $fileKey . "Uploaded successfully ";
  }
}
```

Listing 1.11: Checking each file uploaded for errors

- A proper file upload script will therefore check each uploaded file by checking the various error codes as shown in listing 1.11.

File Size Restrictions

- Some scripts limit the file size of each upload.
- There are many reasons to do so, and ideally we would prevent the file from even being transmitted in the first place if it is too large.
- There are three main mechanisms for maintaining uploaded file size restrictions:
    - **via HTML in the input form**,
        - This technique allows your php.ini maximum file size to be large, while letting some forms override that large limit with a smaller one

```
<form enctype='multipart/form-data' method='post'>
   <input type="hidden" name="MAX_FILE_SIZE" value="1000000" />
   <input type='file' name='file1' />
   <input type='submit' />
</form>
```

Listing 1.12: Limiting upload file size via HTML

    - **via JavaScript in the input form, and**
        - As intuitive as it is, this hidden field can easily be overridden by the client, and is therefore unacceptable as the only means of limiting size.

```
<script>
var file = document.getElementById('file1');
var max_size = document.getElementById("max_file_size").value;
if (file.files && file.files.length ==1){
   if (file.files[0].size > max_size) {
      alert("The file must be less than " + (max_size/1024) + "KB");
      e.preventDefault();
   }
}
</script>
```

Listing 1.13: Limiting upload file size via JavaScript

    - **via PHP coding.**
        - This technique checks the file size on the server by simply checking the size field in the $_FILES array.

```
$max_file_size = 10000000;
foreach($_FILES as $fileKey => $fileArray) {
   if ($fileArray["size"] > $max_file_size) {
      echo "Error: " . $fileKey . " is too big";
   }
   printf("%s is %.2f KB", $fileKey, $fileArray["size"]/1024);
}
```

Listing 1.14: Limiting upload file size via PHP

Limiting the Type of File Upload

- Even if the upload was successful and the size was within the appropriate limits, we may still have a problem.
- What if we wanted the user to upload an image and they uploaded a Microsoft Word document? We might also want to limit the uploaded image to certain image types, such as jpg and png, while disallowing bmp and others.
- To accomplish this type of checking you typically examine the file extension and the type field and also to compare the type to valid image types.
- Listing 1.15 shows sample code to check the file extension of a file, and also to compare the type to valid image types.

```
$validExt = array("jpg", "png");
$validMime = array("image/jpeg","image/png");
foreach($_FILES as $fileKey => $fileArray ){
   $extension = end(explode(".", $fileArray["name"]));
   if (in_array($fileArray["type"],$validMime) &&
         in_array($extension, $validExt)) {
      echo "all is well. Extension and mime types valid";
   }
   else {
      echo $fileKey." Has an invalid mime type or extension";
   }
}
```

Listing 1.15: PHP code to look for valid mime types and file extensions

```
$fileToMove = $_FILES['file1']['tmp_name'];
$destination = "./upload/" . $_FILES["file1"]["name"];
if (move_uploaded_file($fileToMove,$destination)) {
   echo "The file was uploaded and moved successfully!";
}
else {
   echo "there was a problem moving the file";
}
```

Listing 1.16: Using move_uploaded_file() function

Moving the File

- With all of our checking completed, we may now finally want to move the temporary file to a permanent location on your server.
- Typically, we make use of the PHP function move_uploaded_file(), which takes in the temporary file location and the file's final destination.
- This function will only work if the source file exists and if the destination location is writable by the web server (Apache).
- If there is a problem the function will return false, and a warning may be output.
- Listing 1.16 illustrates a simple use of the function.

Reading/Writing Files

- Before the age of the ubiquitous database, software relied on storing and accessing data in files.
- In web development, the ability to read and write to text files remains an important technical competency.

- Even if our site uses a database for storing its information, the fact that the PHP file functions can read/write from a file or from an external website (i.e., from a URL) means that file system functions still have relevance even in the age of database-driven websites.
- There are two basic techniques for read/writing files in PHP:
  - **Stream access**. In this technique, our code will read just a small portion of the file at a time. While this does require more careful programming, it is the most memory-efficient approach when reading very large files.
  - **All-In-Memory access**. In this technique, we can read the entire file into memory (i.e., into a PHP variable). While not appropriate for large files, it does make processing of the file extremely easy.

Stream access

- In the C-style file access you separate the acts of opening, reading, and closing a file.
- The function fopen() takes a file location or URL and access mode as parameters.
- Some of the common modes are "r" for read, "rw" for read and write, and "c," which creates a new file for writing.
- Once the file is opened, we can read from it in several ways.
  - To read a single line, use the fgets() function, which will return false if there is no more data,
  - To read an arbitrary amount of data (typically for binary files), use fread() and for reading a single character use fgetsc().
  - Finally, when finished processing the file you must close it using fclose().
- Listing 1.17 illustrates a script using fopen(), fgets(), and fclose() to read a file and echo it out.
- To write data to a file, we can employ the fwrite() function in much the same way as fgets(), passing the file handle and the string to write.

```
$f = fopen("sample.txt", "r");
$ln = 0;
while ($line = fgets($f)) {
    $ln++;
    printf("%2d: ", $ln);
    echo $line . "<br>";
}
fclose($f);
```

Listing 9.19 Opening, reading lines, and closing a file

Function	Description
file()	Reads the entire file into an array, with each array element corresponding to one line in the file
file_get_contents	Reads the entire file into a string variable
file_put_contents	Writes the contents of a string variable out to a file

Table 1.3: In-Memory File Functions

All-In-Memory access

- While the previous approach to reading/writing files gives you complete control, the programming requires more care in dealing with the streams, file handles, and other low-level issues.
- The alternative simpler approach is much easier to use, at the cost of relinquishing fine-grained control as shown in table 1.3.
- The file_get_contents() and file_put_contents() functions allow you to read or write an entire file in one function call. To read an entire file into a variable you can simply use:

```
$fileAsString = file_get_contents(FILENAME);
```

To write the contents of a string $writeme to a file, you use

```
file_put_contents(FILENAME, $writeme);
```

- These functions are especially convenient when used in conjunction with PHP's many powerful string-processing functions.

Chapter 2: PHP Classes and Objects

1. Object-Oriented Overview
2. Classes and Objects in PHP
3. Object Oriented Design

Object-Oriented Overview

Terminology

- The notion of programming with objects allows the developer to think about an item with particular properties (attributes /data members) and methods (functions).
- The structure of these objects are defined by classes, which outline the properties and methods like a blueprint.
- Each variable created from a class is called an object or instance, and each object maintains its own set of variables, and behaves independently from the class once created.

- Figure 2.1 illustrates the differences between a class, which defines an object's properties and methods, and the objects or instances of that class.

Figure 2.1: Relationship between a class and its objects

The Unified Modeling Language

- The standard diagramming notation for object-oriented design is UML (Unified Modeling Language).
- UML is a succinct set of graphical techniques to describe software design which gives the visualization / representation of objects and classes.
- Several types of UML diagram are defined. Class diagrams and object diagrams, in particular, are useful to us when describing the properties, methods, and relationships between classes and objects.
- **Ex:** To illustrate classes and objects in UML, consider the artist we have looked at in the Art Case Study. Every artist has a first name, last name, birth date, birth city, and death date. Using objects we can encapsulate those properties together into a class definition for an Artist.

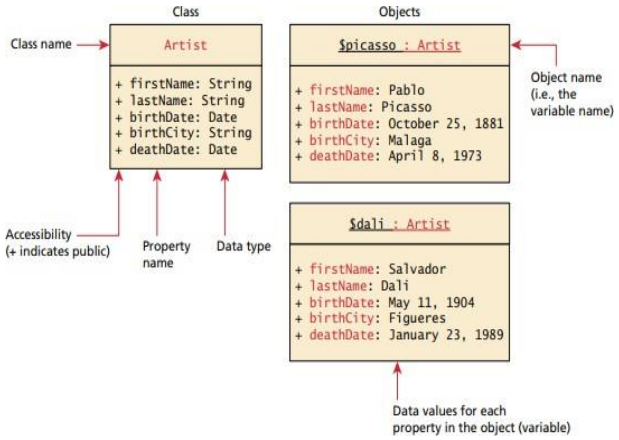


Figure 2.2: Relationship between a class and its objects in UML

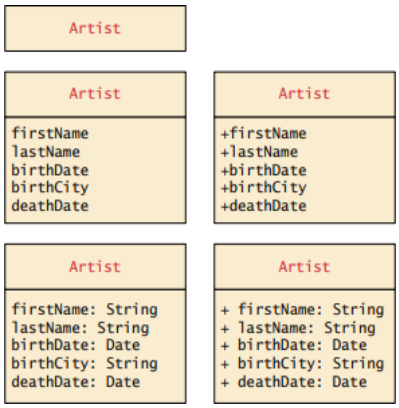


Figure 2.3: Different levels of UML detail

Differences between Server and Desktop Objects

- One important distinction between web programming and desktop application programming is that the objects you create (normally) only exist until a web script is terminated.
- While desktop software can load an object into memory and make use of it for several user interactions, a PHP object is loaded into memory only for the life of that HTTP request.
- Figure 10.4 shows an illustration of the lifetimes of objects in memory between a desktop and a browser application.
- For this reason, we must use classes differently than in the desktop world, since the object must be recreated and loaded into memory for each request that requires it.
- Object-oriented web applications can see significant performance degradation compared to their functional counterparts if objects are not utilized correctly.
- Unlike a desktop, there are potentially many thousands of users making requests at once, so not only objects are destroyed upon responding to each request, but memory must be shared between many simultaneous requests, each of which may load objects into memory.

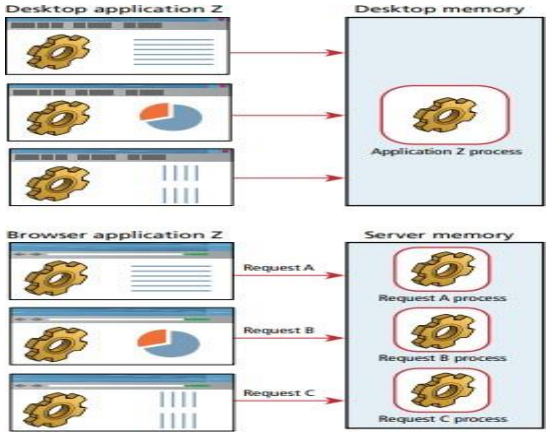


Figure 2.4: Lifetime of objects in memory in web versus desktop applications

Classes and Objects in PHP

- Classes should be defined in their own files so they can be imported into multiple scripts.
- Any PHP script can make use of an external class by using one of the include statements or functions, that is, include, include_once, require, or require_once.
- Once a class has been defined, we can create as many instances of that object as memory will allow using the new keyword.

Defining Classes

- The PHP syntax for defining a class uses the class keyword followed by the classname and { } braces.
- The properties and methods of the class are defined within the braces.
- The Artist class with the properties illustrated in Figure 2.2 is defined using PHP in Listing 2.1.
- Each property in the class is declared using one of the keywords public, protected, or private followed by the property or variable name.

```
class Artist {
    public    $firstName;
    public    $lastName;
    public    $birthDate;
    public    $birthCity;
    public    $deathDate;
}
```

```
$picasso = new Artist();
$dali = new Artist();
$picasso->firstName = "Pablo";
$picasso->lastName = "Picasso";
$picasso->birthCity = "Malaga";
$picasso->birthDate = "October 25 1881";
$picasso->deathDate = "April 8 1973";
```

Listing 2.1: Artist class **Listing 2.2** Instantiating two Artist objects and setting its object's properties

Instantiating Objects

- It's important to note that defining a class is not the same as using it.
- To make use of a class, one must **instantiate** (create) objects from its definition using the new keyword.
- To create two new instances of the Artist class called $picasso and $dali, we instantiate two new objects using the new keyword as follows:

```
$picasso = new Artist();
$dali = new Artist();
```

Properties

- Once we have instances of an object, we can access and modify the properties of each one separately using the variable name and an arrow (→), which is constructed from the dash and greater than symbols.
- Listing 2.2 Shows code that defines the two Artist objects and then sets all the properties for the $picasso object.

Constructors

- Listing 2.2 takes multiple lines and every line of code introduces potential maintainability problems, especially when we define more artists.
- Inside of a class definition, we should therefore define constructors.
- In PHP, constructors are defined as functions with the name __construct().

- Listing 2.3 shows an updated Artist class definition that now includes a constructor.
- Inside of a class you must always use the $this syntax to reference all properties and methods associated with this particular instance of a class.

```
class Artist {
   // variables from previous listing still go here
   ...

   function __construct($firstName, $lastName, $city, $birth,
                        $death=null) {
      $this->firstName = $firstName;
      $this->lastName = $lastName;
      $this->birthCity = $city;
      $this->birthDate = $birth;
      $this->deathDate = $death;
   }
}
```

Listing 2.3: A constructor added to the class definition

```
class Artist {
   ...
   public function outputAsTable() {
      $table = "<table>";
      $table .= "<tr><th colspan='2'>";
      $table .= $this->firstName . " " . $this->lastName;
      $table .= "</th></tr>";
      $table .= "<tr><td>Birth:</td>";
      $table .= "<td>" . $this->birthDate;
      $table .= "(" . $this->birthCity . ")</td></tr>";
      $table .= "<tr><td>Death:</td>";
      $table .= "<td>" . $this->deathDate . "</td></tr>";
      $table .= "</table>";
      return $table;
   }
}
```

Listing 2.4: Method definition

- This new constructor can then be used when instantiating as shown below

```
$picasso = new Artist("Pablo","Picasso","Malaga","Oct 25,1881",
                      "Apr 8,1973");
$dali = new Artist("Salvador","Dali","Figures","May 11 1904",
                   "Jan 23 1989");
```

Methods

- Objects are useful when we define behavior or operations that they can perform.
- In object-oriented lingo these operations are called methods and are like functions, except they are associated with a class as shown in listing 2.4.
- They define the tasks each instance of a class can perform and are useful since they associate behavior with objects.
- To output the artist, you can use the reference and method name as follows:

```
$picasso = new Artist( . . . )
echo $picasso->outputAsTable();
```

- Updated UML diagram is shown below.

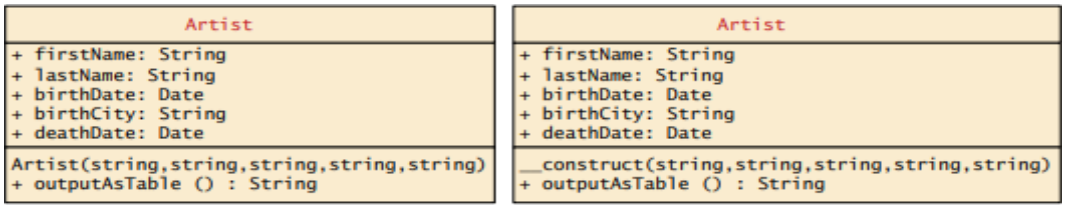


Figure 2.5: Updated class diagram

Visibility

- The visibility of a property or method determines the accessibility of a class member (i.e., a property or method) and can be set to public, private, or protected.

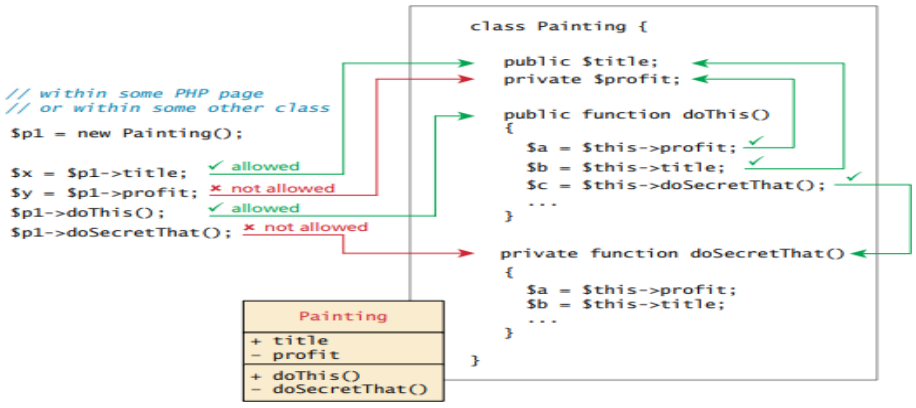


Figure 2.6: Visibility of class members

- **Public**
  - The public keyword means that the property or method is accessible to any code that has a reference to the object.
- **Private**
  - The private keyword sets a method or variable to only be accessible from within the class.
  - This means that we cannot access or modify the property from outside of the class even if we have a reference to it as shown in Figure 2.6.
- **Protected**
  - Protected data members and methods are only accessible by the classes of the same package and the subclasses present in any package.
- In UML, the "+" symbol is used to denote public properties and methods, the "–" symbol for private ones, and the "#" symbol for protected ones.

Static Members

- A static member is a property or method that all instances of a class share and there is only one value for a class's static property.
- To illustrate how a static member is shared between instances of a class, we will add the static property artistCount to our Artist class, and use it to keep a count of how many Artist objects are currently instantiated.

- This variable is declared static by including the static keyword in the declaration as shown in listing 2.5

```
class Artist {
    public static $artistCount = 0;
    public   $firstName;
    public   $lastName;
    public   $birthDate;
    public   $birthCity;
    public   $deathDate;

    function __construct($firstName, $lastName, $city, $birth,
                         $death=null) {
       $this->firstName = $firstName;
       $this->lastName = $lastName;
       $this->birthCity = $city;
       $this->birthDate = $birth;
       $this->deathDate = $death;
       self::$artistCount++;
    }
 }
```

Listing 2.5: Class definition modified with static members

- Notice that we do not reference a static property using the $this-> syntax, but rather it has its own self:: syntax.
- The rationale behind this change is to force the programmer to understand that the variable is static and not associated with an instance ($this).
- This static variable can also be accessed without any instance of an Artist object by using the class name, that is, via Artist::$artistCount.
- Static methods are similar to static properties in that they are globally accessible (if public) and are not associated with particular objects.
- It should be noted that static methods cannot access instance members.
- Static methods are called using the same double colon syntax as static properties.

Class Constants

- To add a property to a class that is constant as same as static members but with a const keyword.

```
const EARLIEST_DATE = 'January 1, 1200';
```

- However, constant values can be stored more efficiently as class constants so long as they are not calculated or updated.

- Unlike all other variables, constants don't use the $ symbol when declaring or using them.
- They can be accessed both inside and outside the class using self::EARLIEST_DATE in the class and classReference::EARLIEST_DATE outside.

Object-Oriented Design

- With the basic understanding of how to define and use classes and objects, we can start to get the benefits of software engineering patterns, which encourage understandable and less error-prone code.
- The object-oriented design of software offers many benefits in terms of modularity, testability, and reusability.

Data Encapsulation

- The most important advantage to object-oriented design is the possibility of **encapsulation**, which generally refers to restricting access to an object's internal components.
- Another way of understanding encapsulation is: it is the hiding of an object's implementation details.
- A properly encapsulated class will define an interface to the world in the form of its public methods, and leave its data, that is, its properties, hidden (that is, private).
- If a properly encapsulated class makes its properties private, then the typical approach is to write methods for accessing and modifying properties rather than allowing them to be accessed directly.
- These methods are commonly called **getters and setters** (or accessors and mutators).
- A **getter** to return a variable's value is often very straightforward and should not modify the property.
- It is normally called without parameters, and returns the property from within the class.
- **Setter** methods modify properties, and allow extra logic to be added to prevent properties from being set to strange values.

- **Ex:**

```
public function getFirstName() {
   return $this->firstName;
}
```

getter function : get()

```
public function setBirthDate($birthdate){
   // set variable only if passed a valid date string
   $date = date_create($birthdate);

   if ( ! $date ) {
      $this->birthDate = $this->getEarliestAllowedDate();
   }
   else {
      // if very early date then change it to
      // the earliest allowed date
      if ( $date < $this->getEarliestAllowedDate() ) {
         $date = $this->getEarliestAllowedDate();
      }
      $this->birthDate = $date;
   }
}
```

setter function: set()

- The key advantage of using getters and setters are:
  - The class can handle the responsibility of ensuring its own data validation.
  - And since the setter functions are performing validation, the constructor for the class should use the setter functions to set the values.

Inheritance

- Along with encapsulation, **inheritance** is one of the three key concepts in object oriented design and programming.
- Inheritance enables us to create new PHP classes that reuse, extend, and modify the behavior that is defined in another PHP class.
- PHP only allows you to inherit from one class at a time.
- A class that is inheriting from another class is said to be a **subclass** or a **derived class**.
- The class that is being inherited from is typically called a **superclass** or a **base class**.
- When a class inherits from another class, it inherits all of its public and protected methods and properties. Figure 2.7 illustrates how inheritance is shown in a UML class diagram.

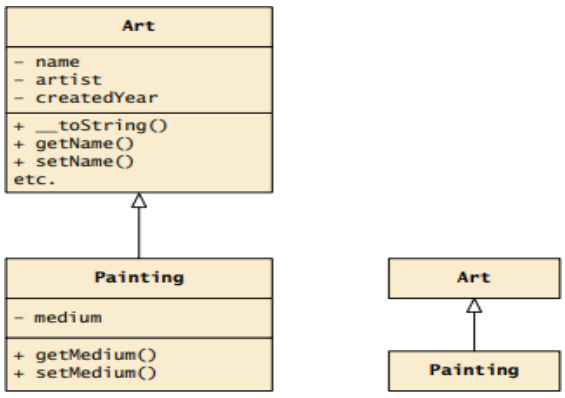


Figure 2.7: UML class diagram with inheritance

- Just as in Java, a PHP class is defined as a subclass by using the extends keyword.

```
class Painting extends Art { . . . }
```

Referencing Base Class Members

- A subclass inherits the public and protected members of the base class. Thus in the following code based on Figure 2.7, both of the references will work because it is as if the base class public members are defined within the subclass.

```
$p = new Painting();
. . .
// these references are ok
echo $p->getName();      // defined in base class
echo $p->getMedium();    // defined in subclass
```

Inheriting Methods

- Every method defined in the base/parent class can be overridden when extending a class, by declaring a function with the same name.
- To access a public or protected method or property defined within a base class from within a subclass, we do so by prefixing the member name with parent::. So to access the parent's __toString() method we would simply use parent::__toString().

Parent Constructors

- If we want to invoke a parent constructor in the derived class's constructor, we can use the parent:: syntax and call the constructor on the first line parent:: __construct().
- This is similar to calling other parent methods, except that to use it we must call it at the beginning of our constructor.

Polymorphism

- Polymorphism is the third key object-oriented concept (along with encapsulation and inheritance).
- Polymorphism is the notion that an object can in fact be multiple things at the same time.
- Let us begin with an instance of a Painting object named $guernica created as follows:

```
$guernica = new Painting("1937",$picasso,"Guernica","Oil on canvas");
```

- The variable $guernica is both a Painting object and an Art object due to its inheritance.
- The advantage of polymorphism is that we can manage a list of Art objects, and call the same overridden method on each.

Object Interfaces

- An object interface is a way of defining a formal list of methods that a class must implement without specifying their implementation.
- Interfaces provide a mechanism for defining what a class can do without specifying how it does it, which is often a very useful design technique.
- Interfaces are defined using the interface keyword, and look similar to standard PHP classes, except an interface contains no properties and its methods do not have method bodies defined.
- Ex:

```
interface Viewable {
    public function getSize();
    public function getPNG();
}
```

- An interface contains only public methods, and instead of having a method body, each method is terminated with a semicolon.
- In PHP, a class can be said to implement an interface, using the implements keyword.

```
class Painting extends Art implements Viewable { ... }
```

- This means then that the class Painting must provide implementations (i.e., normal method bodies) for the getSize() and getPNG() methods.

Chapter 3: Error Handling and Validation

1. What are Errors and Exceptions?
2. PHP Error Reporting
3. PHP Error and Exception Handling

What Are Errors and Exceptions?

- Even the best-written web application can suffer from runtime errors.
- Most complex web applications must interact with external systems such as databases, web services, RSS feeds, email servers, file system, and other externalities that are beyond the developer's control.
- A failure in any one of these systems will mean that the web application will no longer run successfully. It is vitally important that web applications gracefully handle such problems.

Types of Errors

- Not every problem is unexpected or catastrophic. One might say that there are three different types of website problems:
  - Expected errors
  - Warnings
  - Fatal errors

- An **expected error** is an error that routinely occurs during an application. Perhaps the most common example of this type would be an error as a result of user inputs, for instance, entering letters when numbers were expected.
- Another type of error is **warnings**, which are problems that generate a PHP warning message (which may or may not be displayed) but will not halt the execution of the page. For instance, calling a function without a required parameter will generate a warning message but not stop execution.
- The final type of error is **fatal errors**, which are serious in that the execution of the page will terminate unless handled in some way. These should truly be exceptional and unexpected, such as a required input file being missing or a database table or field disappearing.

Exceptions

- Developers sometimes treat the words "error" and "exception" as synonyms. In the context of PHP, they do have different meanings.
- An **error** is some type of problem that generates a nonfatal warning message or that generates an error message that terminates the program's execution.
- An **exception** refers to objects that are of type Exception and which are used in conjunction with the object-oriented try . . . catch language construct for dealing with runtime errors.

PHP Error Reporting

- PHP has a flexible and customizable system for reporting warnings and errors that can be set programmatically at runtime or declaratively at design-time within the **php.ini.**
- There are three main error reporting flags:
  - error_reporting
  - display_errors
  - log_errors
- The meaning of each of these is important and should be learned by PHP developers.

The error_reporting Setting

- The error_reporting setting specifies which type of errors are to be reported.
- It can be set programmatically inside any PHP file by using the error_reporting()function:

```
error_reporting(E_ALL);
It can also be set within the php.ini file:
error_reporting = E_ALL
```

- The possible levels for error_reporting are defined by predefined constants lists some of the most common values. It is worth noting that in some PHP environments, the default setting is zero, that is, no reporting.

The display_errors Setting

- The display_error setting specifies whether error messages should or should not bedisplayed in the browser.2 It can be set programmatically via the ini_set() function:

```
ini_set('display_errors','0');
It can also be set within the php.ini file:
display_errors = Off
```

The log_error setting

- The log_error setting specifies whether error messages should or should not be sent to the server error log. It can be set programmatically via the ini_set() function:

```
ini_set('log_errors','1');
It can also be set within the php.ini file:
log_errors = On
```

PHP Error and Exception Handling

- When a fatal PHP error occurs, program execution will eventually terminate unless it is handled.
- The PHP documentation provides two mechanisms for handling runtime errors: procedural error handling and the more object-oriented exception handling.

Procedural Error Handling

- In the procedural approach to error handling, the programmer needs to explicitly test for error conditions after performing a task that might generate an error.
- In such a case we needed to test for and deal with errors after each operation that might generate an error state.
- While this approach might seem more straightforward, it does require the programmer to know ahead of time what code is going to generate an error condition.
- The advantage of the try . . . catch mechanism is that it allows the developer to handle a wider variety of exceptions in a single catch block.
- Even with explicit testing for error conditions, there will still be situations when an unforeseen error occurs. In such a case, unless a custom error handler has been defined, PHP will terminate the execution of the application.

```
$connection = mysqli_connect(DBHOST, DBUSER, DBPASS, DBNAME);

$error = mysqli_connect_error();
if ($error != null) {
   // handle the error
   ...
}
```

Listing 3.1: Procedural approach to error handling

Object-Oriented Exception Handling

- When a runtime error occurs, PHP *throws* an *exception*.
- This exception can be *caught* and handled either by the function, class, or page that generated the exception or by the code that called the function or class.
- If an exception is not caught, then eventually the PHP environment will handle it by terminating execution with an "Uncaught Exception".
- Like other object-oriented programming languages, PHP uses the try .. .catch programming construct to programmatically deal with exceptions at runtime.
- The Exception class provides methods for accessing not only the exception message, but also the line number of the code that generated the exception and the stack trace, both of which can be helpful for understanding where and when the exception occurred.

```
// Exception throwing function
function throwException($message = null,$code = null) {
  throw new Exception($message,$code);
}

try {
  // PHP code here
  $connection = mysqli_connect(DBHOST, DBUSER, DBPASS, DBNAME)
    or throwException("error");
 //...
}
catch (Exception $e) {
  echo ' Caught exception: ' .  $e->getMessage();
  echo ' On Line : ' .  $e->getLine();
  echo ' Stack Trace: '; print_r($e->getTrace());
} finally {
  // PHP code here that will be executed after try or after catch
}
```

Listing 3.2: Example of try . . . catch block

- The finally block is optional. Any code within it will always be executed after the code in the try or in the catch blocks, even if that code contains a return statement.

Custom Error and Exception Handlers

- When a web application is in development, one can generally be content with displaying and/or logging error messages and then terminating the script.
- But for production applications, we will likely want to handle significant errors in a better way.

- It is possible to define our own handler for uncaught errors and exceptions; the mechanism for doing so varies depending upon whether you are using the procedural or object-oriented mechanism for responding to errors.
- If using the procedural approach (i.e., *not* using try . . . catch), we can define a custom *error*-handling function and then register it with the set_error_handler() function.
- If we are using the object-oriented exception approach with try . . . catch blocks, we can define a custom *exception*-handling function and then register it with the set_exception_handler() function.

```
function my_exception_handler($exception) {
  // put together a detailed exception message
  $msg = "<p>Exception Number " . $exception->getCode();
  $msg .= $exception->getMessage() . " occurred on line ";
  $msg .= "<strong>" . $exception->getLine() . "</strong>";
  $msg .= " and in the file: ";
  $msg .= "<strong>" . $exception->getFile() . "</strong> </p>";

  // email error message to someone who cares about such things
  error_log($msg, 1, 'support@domain.com',
            'From: reporting@domain.com');
  // if exception serious then stop execution and tell maintenance fib
  if ($exception->getCode() !== E_NOTICE) {
     die("Sorry the system is down for maintenance. Please try
          again soon");
  }
}
```

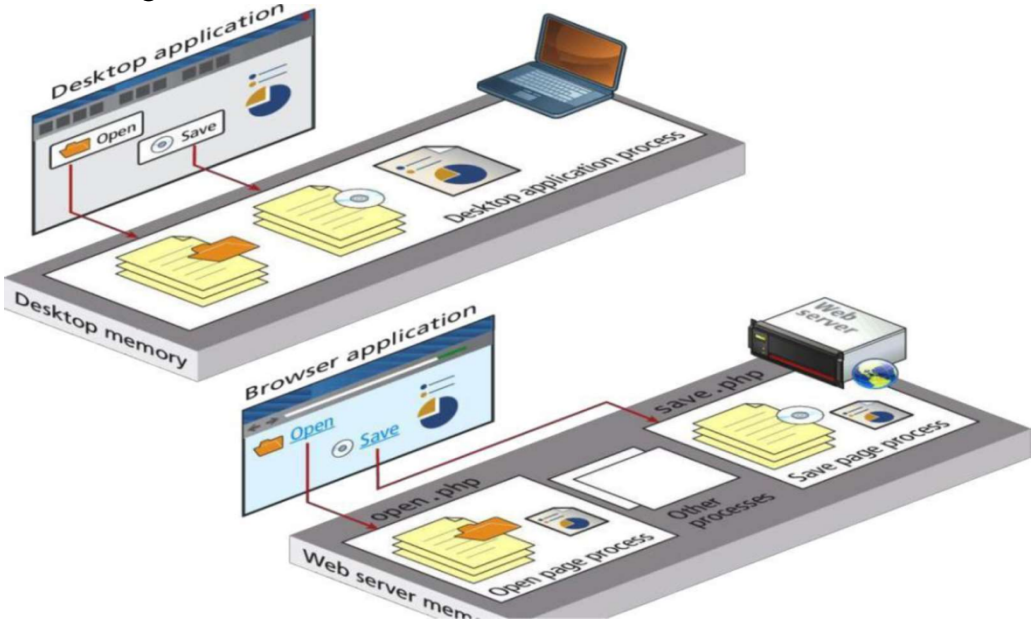
Listing 3.3: Custom exception handler

MODULE – 5

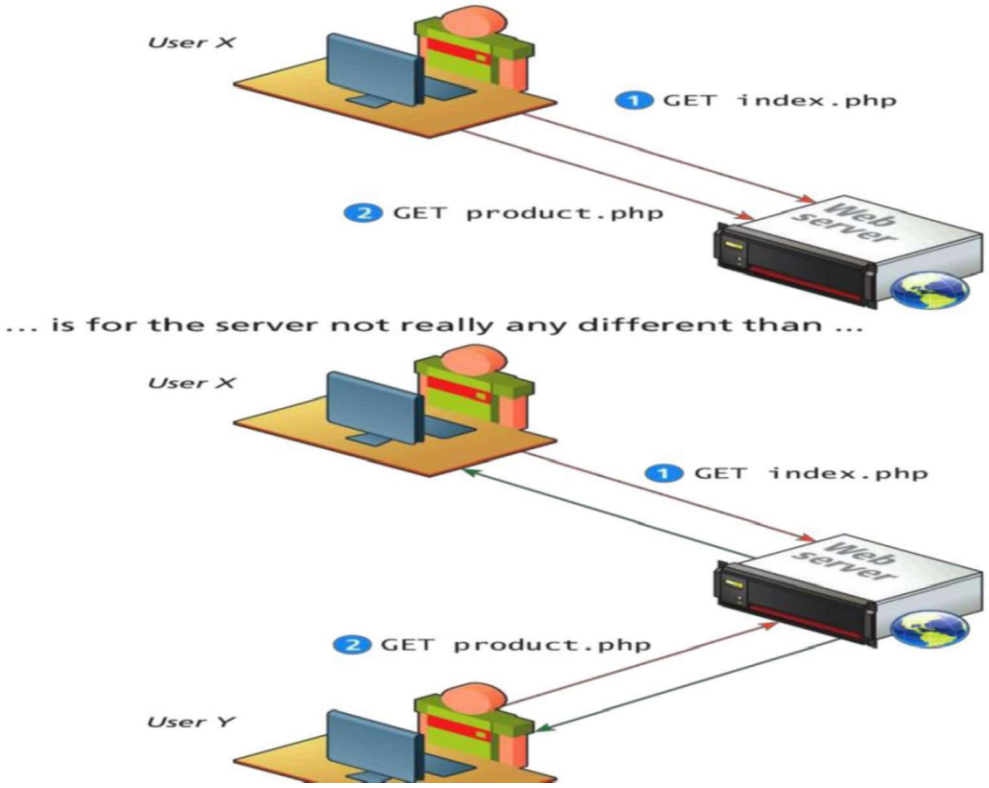
MANAGING STATES

THE PROBLEM OF STATE IN WEB APPLICATIONS

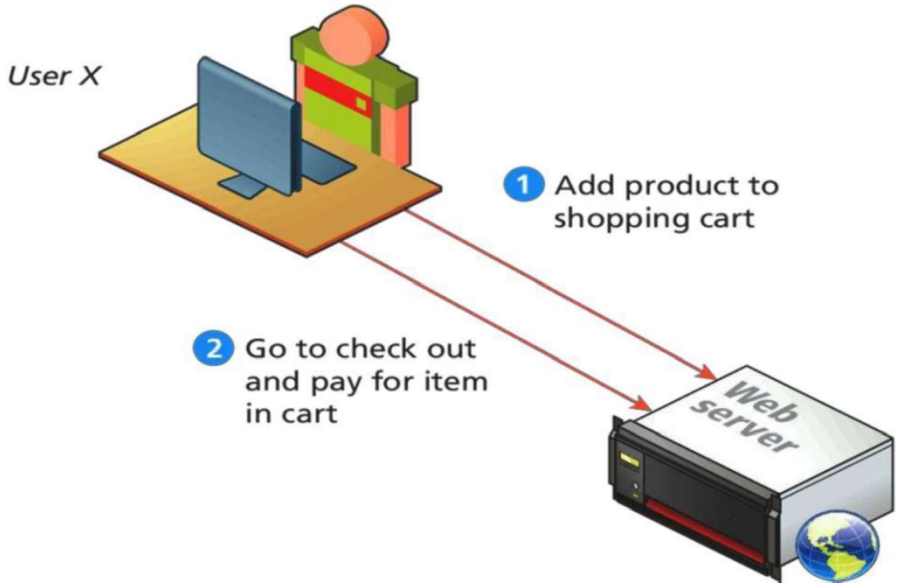
- Until now we have seen how to process user inputs, output information and read & write from other storage media.
- But here we will be examining a development problem that is unique to the world of web development.
- Single user desktop applications do not have this challenge at all because the program information for the user is stored in memory (or in external storage) and thus can be easily accesses throughout the applications.
  - Remember Web applications differ from desktop applications
- Unlike the unified single process that is the typical desktop application, a web application consists of a series of disconnected HTTP requests to a web server where each request for a server page is essentially a request to run a separate program as in below fig



- The web server sees only requests.
- The HTTP protocol does not, without program intervention, distinguish two requests by one source from two requests from two different sources, as in below figure



- There are many occasions when we want the web server to connect requests together.
- Consider the scenario of a web shopping cart, as in below fig.
- In such a case, the user(website owner) most certainly wants the server to recognize that the request to add an item to the cart and the subsequent request to check out and pay for the item in the cart are connected to the same individual



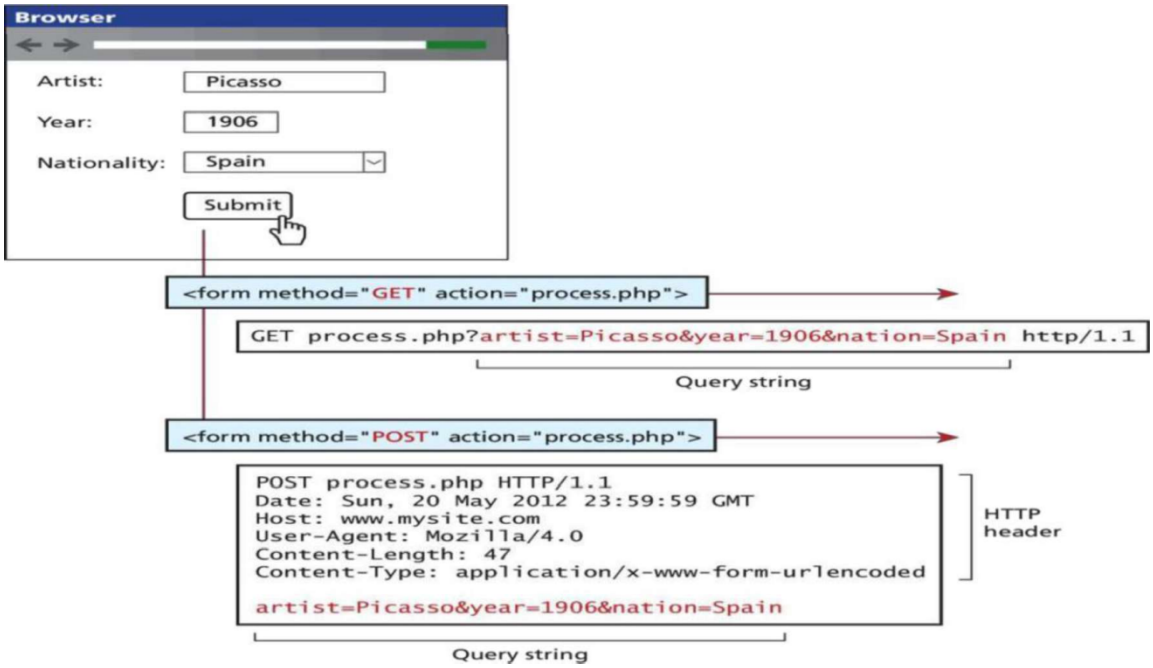
- The rest of the chapter we will explain how web developers and web development environments work together through the constraints of HTTP to solve this particular problem.
- How does one web page pass information to another page?
- What mechanisms are available within HTTP to pass information to the server in our requests?

In HTTP, we can pass information using:

- Query strings
- Cookies

PASSINGINFORMATION VIA QUERY STRINGS

- A web page can pass query string information from the browser to the server using one of the two methods:
  - A query string within the URL(GET)
  - A query string within the HTTP header(POST)



PASSINGINFORMATION VIA THE URL PATH

- Passing information from one page to another is done by query strings but they have drawback.
- The URLs that result can be long and complicated.
- while there is some dispute about whether dynamic URLs(i.e., ones with query string parameters) or static URLs are better from a search engine result optimization (or SEO)

- Dynamic URLs (i.e., query string parameters) are a pretty essential part of web application development.
- How can we do without them?
- The answer is to rewrite the dynamic URL into a static one (and vice versa). This process is commonly called **URL rewriting**.
- In below fig, the top four commerce – related results for the search term "Reproduction Raphael portrait Donna velata" are shown along with their URLs.
- Notice how the top three do not use query string parameters but instead put the relevant information within the folder path or the file name.

URL rewriting

- We can try doing our own rewriting. Let us begin with the following URL with its query string information:

www.somedomain.com/DisplayArtist.php?artist=16

- One typical alternate approach would be to rewrite the URL to:

www.somedomain.com/artists/16.php

- Notice that the query string name and value have been turned into path names. One could improve this to make it more SEO friendly using the following:

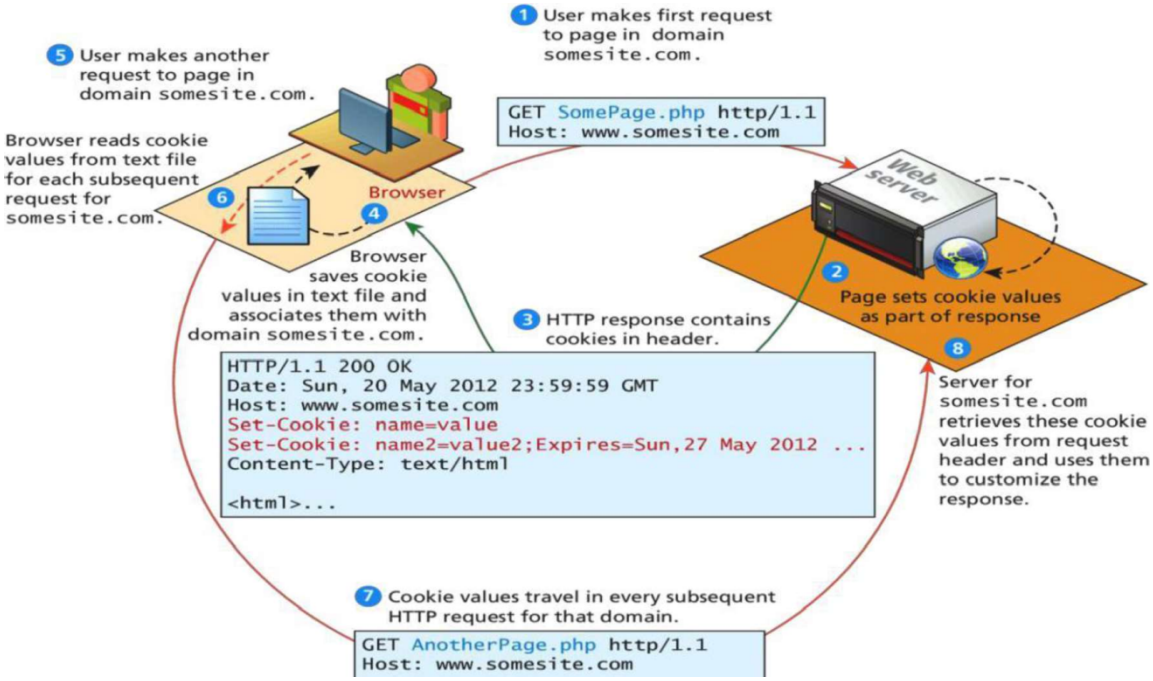
www.somedomain.com/artists/Mary-Cassatt

URL rewriting in Apache and linux

- The mod_rewrite module uses a rule-based rewriting engine that utilizes Perl compatible regular expressions to change the URLs so that the requested URL can be mapped or redirected to another URL internally.

COOKIES

- **Cookies** are a client-side approach for persisting state information.
- They are name=value pairs that are saved within one or more text files that are managed by the browser.
- While cookie information is stored and retrieved by the browser, the information in a cookie travels within the HTTP header.
- Sites that use cookies should not depend on their availability for critical features
- The user can delete cookies or tamper with them



Two kinds of Cookie

- A **session cookie** has no expiry stated and thus will be deleted at the end of the user browsing session.
- **Persistent cookies** have an expiry date specified;

```
<?php
    // add 1 day to the current time for expiry time
    $expiryTime = time()+60*60*24;

    // create a persistent cookie
    $name = "Username";
    $value = "Ricardo";
    setcookie($name, $value, $expiryTime);
?>
```

LISTING 13.1 Writing a cookie

```php
<?php
   if( !isset($_COOKIE['Username']) ) {
      //no valid cookie found
   }
   else {
      echo "The username retrieved from the cookie is:";
      echo $_COOKIE['Username'];
   }
?>
```

LISTING 13.2 Reading a cookie

- In addition to being used to track authenticated users and shopping carts, cookies can implement:

- "Remember me" persistent cookie
- Store user preferences
- Track a user"s browsing behavior

SERIALIZATION

- **Serialization** is the process of taking a complicated object and reducing it down to zeros and ones for either storage or transmission.
- In PHP objects can easily be reduced down to a binary string using the **serialize ()** function.
- The string can be reconstituted back into an object using the **unserialize()** method

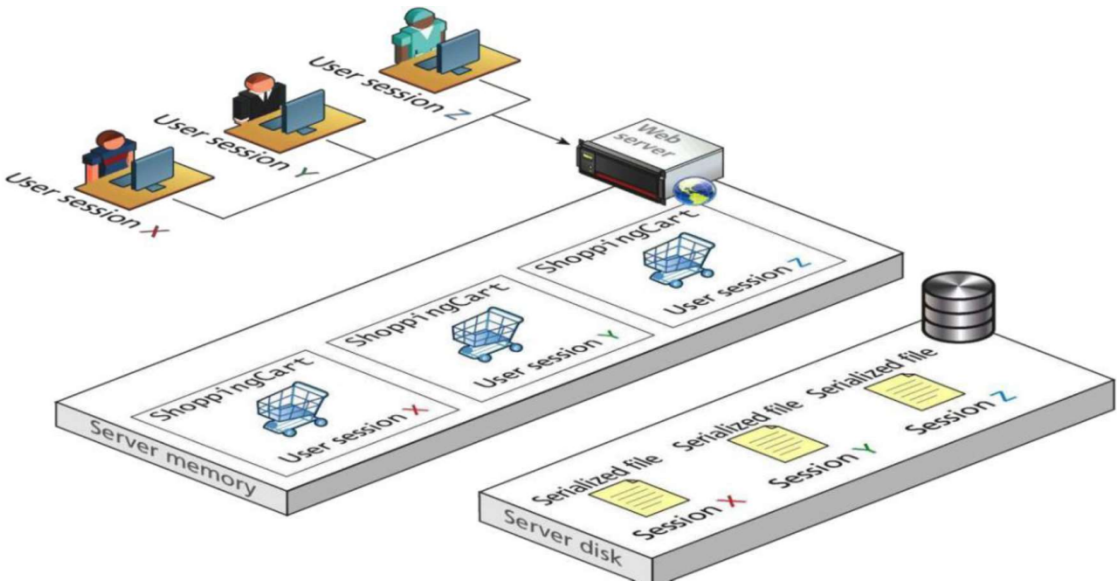
```php
interface Serializable {
   /* Methods */
   public function serialize();
   public function unserialize($serialized);
}
```

LISTING 13.3 The Serializable interface

Application of Serialization

- Since each request from the user requires objects to be reconstituted, using serialization to store and retrieve objects can be a rapid way to maintain state between requests.
- At the end of a request you store the state in a serialized form, and then the next request would begin by deserializing it to re establish the previous state.

SESSIONSTATE



- All modern web development environments provide some type of session state mechanism.
- **Session state** is a server-based state mechanism that lets web applications store and retrieve objects of any type for each unique user session.
- Session state is ideal for storing more complex objects or data structures that are associated with a user session.
- In PHP, session state is available to the via the $_SESSION variable
- Must use session_start() to enable sessions.

```php
<?php

session_start();

if ( isset($_SESSION['user']) ) {
    // User is logged in
}
else {
    // No one is logged in (guest)
}
?>
```

LISTING 13.5 Accessing session state

```php
<?php
include_once("ShoppingCart.class.php");

session_start();

// always check for existence of session object before accessing it
if ( !isset($_SESSION["Cart"]) ) {
   //session variables can be strings, arrays, or objects, but
   // smaller is better
   $_SESSION["Cart"] = new ShoppingCart();
}
$cart = $_SESSION["Cart"];
?>
```

LISTING 13.6 Checking session existence

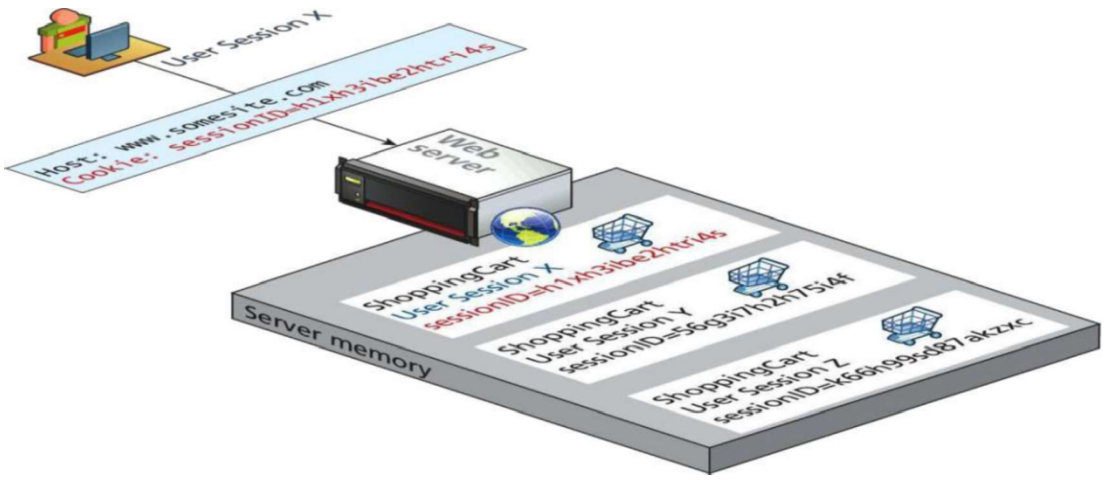

```php
<?php
include_once("ShoppingCart.class.php");

session_start();

// always check for existence of session object before accessing it
if ( !isset($_SESSION["Cart"]) ) {
   //session variables can be strings, arrays, or objects, but
   // smaller is better
   $_SESSION["Cart"] = new ShoppingCart();
}
$cart = $_SESSION["Cart"];
?>
```

LISTING 13.6 Checking session existence

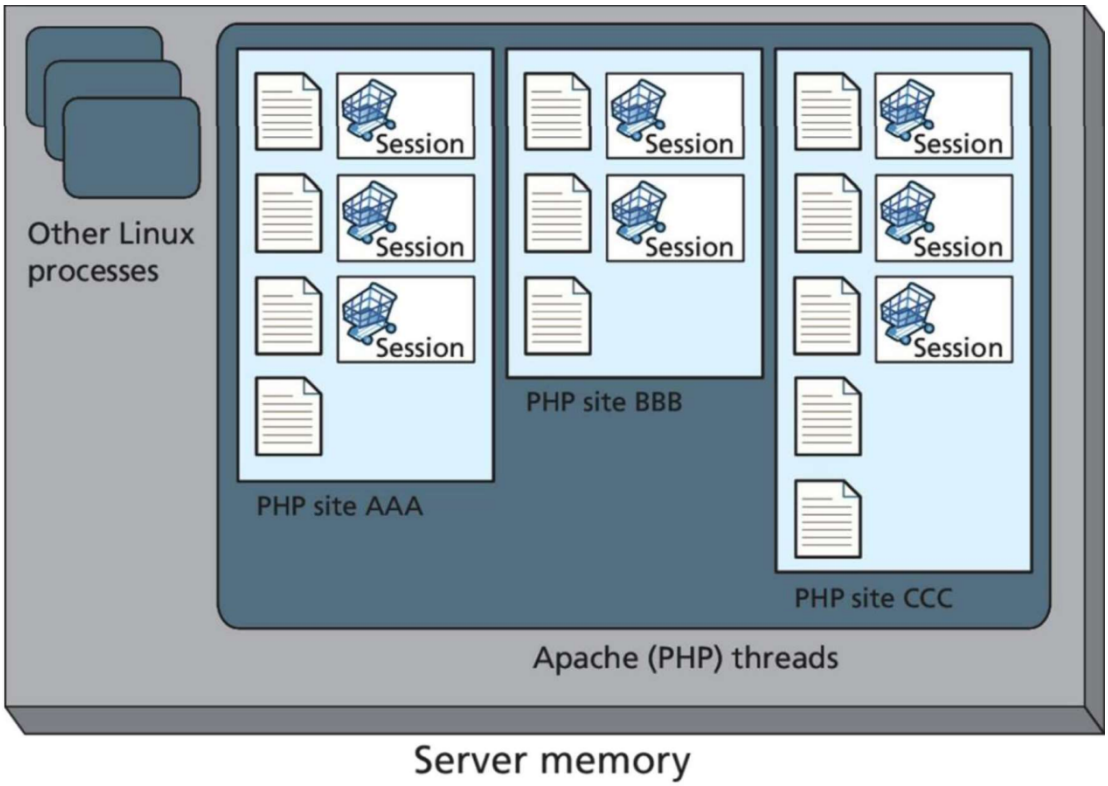
How does state session work?



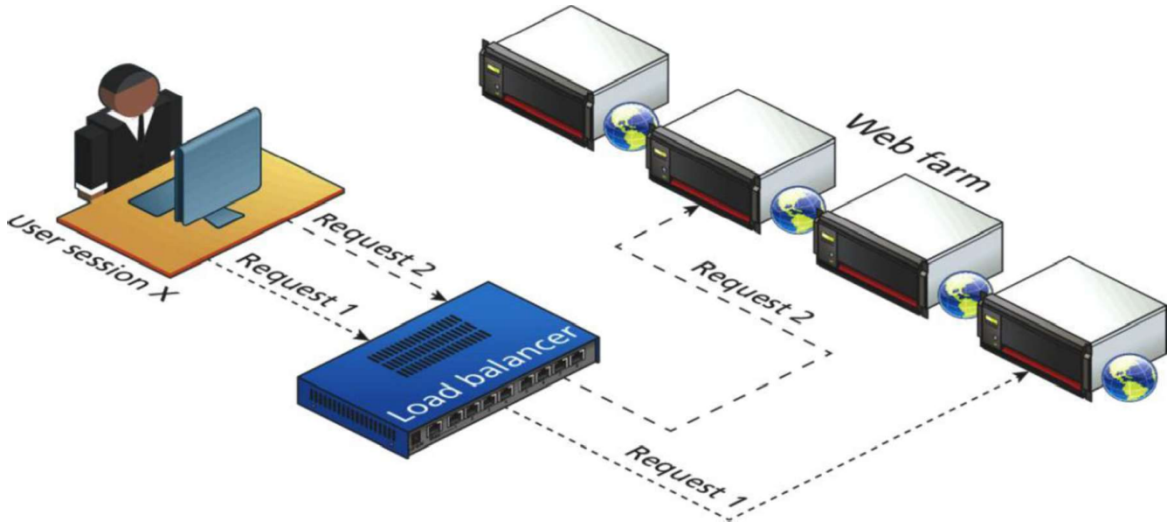
- Sessions in PHP are identified with a unique 32- byte session ID.
- This is transmitted back and forth between the user and the server via a session cookie.
- For a brand new session, PHP assigns an initially empty dictionary-style collection that can be used to hold any state values for this session.
- When the request processing is finished, the session state is saved to some type of state storage mechanism, called a session state provider
- When a new request is received for an already existing session, the session's dictionary collection is filled with the previously saved session data from the session state provider.

Session Storage

- It is possible to configure many aspects of sessions including where the session files are saved.
- The decision to save sessions to files rather than in memory (like ASP.NET) addresses the issue of memory usage that can occur on shared hosts as well as persistence between restarts.
- Inexpensive web hosts may sometimes stuff hundreds or even thousands of sites on each machine.
- Server memory may be storing not only session information, but pages being executed, and caching information.

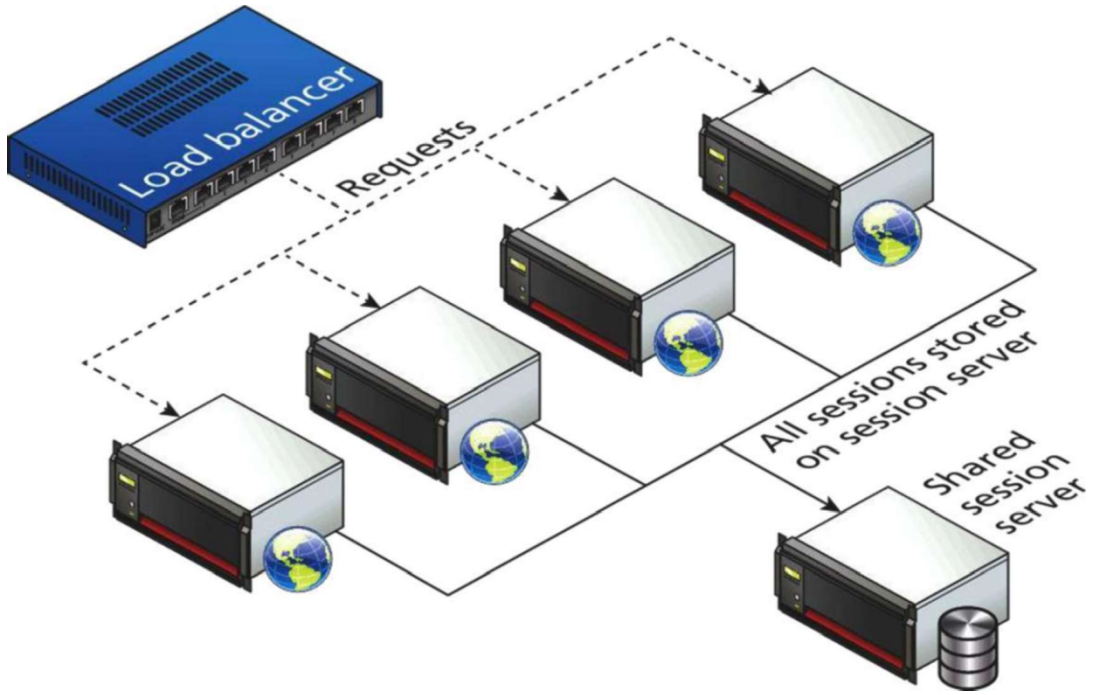


- Higher-volume web applications often run in an environment in which multiple web servers (also called a web farm) are servicing requests.
- In such a situation the in-process session state will not work, since one server may service one request for a particular session, and then a completely different server may service the next request for that session



There are effectively two categories of solution to this problem.

1. Configure the load balancer to be "session aware" and relate all requests using a session to the same server.
2. Use a shared location to store sessions, either in a database, mem cache, or some other shared session state mechanism



```
[Session]
; Handler used to store/retrieve data.
session.save_handler = memcache
session.save_path = "tcp://sessionServer:11211"
```

LISTING 13.7 Configuration in php.ini to use a shared location for sessions

HTML5 Web Storage

- Web storage is a new JavaScript only API introduced in HTML5.
- IT is meant to be a replacement to cookies, in that web storage is managed by the browser.
- Unlike cookies web storage data is not transported to and from the server with every request and response
- Web storage is not limited to 4k size barrier of cookies.
- Limit of 5 MB but browsers are allowed to store more per domain.
- Just as there were two types of cookies, there are two types of web storage
  - Local storage –It is for saving information that will persist between browser sessions.
  - Session storage – It is for information that will be lost once the browser session is finished.
- Below code illustrates the JavaScript code for writing information to web storage.
- There are two ways to store values in web storage
- Using setItem() function, or using the property shortcut(Eg: session storage.FavoriteArtist().

```
<form ... >
   <h1>Web Storage Writer</h1>
   <script language="javascript" type="text/javascript">

        if (typeof (localStorage) === "undefined" ||
              typeof (sessionStorage) === "undefined") {
            alert("Web Storage is not supported on this browser...");
        }
        else {
            sessionStorage.setItem("TodaysDate", new Date());
            sessionStorage.FavoriteArtist = "Matisse";

            localStorage.UserName = "Ricardo";
            document.write("web storage modified");
        }
   </script>
   <p><a href="WebStorageReader.php">Go to web storage reader</a></p>
</form>
```

LISTING 13.8 Writing web storage

- Below code illustrates the process of reading from web storage is equally straight forward.
- The difference between session Storage and local Storage in this example is that if you close the browser after writing and then run the code (Above code), only the local storage item will still contain a value.

```
<form id="form1" runat="server">
   <h1>Web Storage Reader</h1>
   <script language="javascript" type="text/javascript">

        if (typeof (localStorage) === "undefined" ||
            typeof (sessionStorage) === "undefined") {
            alert("Web Storage is not supported on this browser...");
        }
        else {
            var today = sessionStorage.getItem("TodaysDate");
            var artist = sessionStorage.FavoriteArtist;

            var user = localStorage.UserName;
            document.write("date saved=" + today);
            document.write("<br/>favorite artist=" + artist);
            document.write("<br/>user name = " + user);
        }
   </script>
</form>
```

LISTING 13.9 Reading web storage

- Cookies have the disadvantage of being limited in size, potentially disabled by the user, other security attacks and being sent in every single request and response to and from a given domain.
- Web storage is not as a cookie replacement but as a local cache for relatively static items available to JavaScript.
- One use of web storage is to store static content downloaded asynchronously such as xml or JSON from a web service in web storage, this reducing server load for subsequent requests by the session.
- Below fig illustrates an example of how web storage could be used as a mechanism for reducing server data request, there by speeding up the display of the page on the browser as well as reducing load on the server.

CACHING

- Caching is the vital way to improve the performance of web applications. Your browser uses caching to speed up the user experience by using locally stored versions of images and other files rather than re-requesting the files from the server.
- A server side developer only had limited control over browser caching.
- Why is this necessary?
- Every time a PHP page is requested, it must be fetched, parsed and executed by the PHP engine and end result is HTML that is sent back to the requestor.

- On way to address this problem is to cache the generated markup in server memory so that subsequent requests can be served from memory rather than from the execution of the page.
- There are two basic strategies to caching web applications.
  - PAGEOUTPUTCACHING

  It saves the rendered output of a page or user control and reuses the output instead of reprocessing the page when a user requests the page again
  - APPLICATIONDATACACHING

  It allows the developer to programmatically cache data.
- There are two models for page caching:
  - Full page caching
  - Partial page caching :

Only specific parts of page are cached while other parts are dynamically generated in the normal manner.

Application of data caching

- One of the biggest drawbacks with page output caching is that performance gains will only be had if the entire cached page is the same for numerous requests.
- An alternate strategy is to use application data caching in which a page will programmatically place commonly used collections of data that require time intensive queries from the database or web server into cache memory, and then other pages that also need that same data can use the cache version rather than re – retrieve it from its original location.
- While the default installation of PHP does not come with an application caching ability, a widely available free PECL extension called memcache is used for this ability.

Advanced JavaScript & JQuery

JAVASCRIPTPSEUDO-CLASSES

- JavaScript has no formal class mechanism, it does support objects (DOM).
- While most OO languages that support objects also support classes formally, JavaScript does not.
- Instead we define Pseudo–classes through a variety of syntax.

Using Object Literals

- An array in JavaScript can be instantiated var daysofweek = [“sun” , ”mon” ,“tue”…..];
- An object can be instantiated using object literals.

- **Object literals** are a list of key-value pairs with colons between the key and value with commas separating key-value pairs.
- Object literals are also known as **Plain Objects** in jQuery.
- JavaScript has no formal class mechanism.

Simple Variable var car ="Fiat";

Then this

var car = {type:"Fiat", model:500, color:"white"};

Properties car.name = Fiat car.model = 500 car.weight = 850kg car.color = white
Methods: car.start() car.drive() car.brake() car.stop()

Emulate Classes with functions

Use functions to encapsulate variables and methods together.

```
function Die(col) {
   this.color=col;
   this.faces=[1,2,3,4,5,6];
}
```

LISTING 15.1 Very simple Die pseudo-class definition as a function

Instantiation looks much like in PHP:

var oneDie = new Die("0000FF");

Adding methods to the objects

- To define a method in an object's function one can either define it internally or use a reference to a function define outside the class.
- One technique for adding a method inside of a class definition is by assigning an anonymous function to a variable

```
function Die(col) {
   this.color=col;
   this.faces=[1,2,3,4,5,6];

   // define method randomRoll as an anonymous function
   this.randomRoll = function() {
      var randNum = Math.floor((Math.random() * this.faces.length)+ 1);
      return faces[randNum-1];
   };
}
```

LISTING 15.2 Die pseudo-class with an internally defined method

Using Prototypes

So you can use a *prototype* of the class.

- Prototypes are used to make JavaScript behave more like an object-oriented language.
- The prototype properties and methods are defined *once* for all instances of an *object*.
- Every object has a prototype

```
// Start Die Class
function Die(col) {
   this.color=col;
   this.faces=[1,2,3,4,5,6];
}

Die.prototype.randomRoll = function() {
   var randNum = Math.floor((Math.random() * this.faces.length) + 1);
   return faces[randNum-1];
};
// End Die Class
```

LISTING 15.3 The Die pseudo-class using the prototype object to define methods

- This definition is better because it defines the method only once, no matter how many instance of die are created.
- How many ever created it will be reference to that one method. (fig below)

JQUERYFOUNDATIONS

- A **library** or **framework** is software that you can utilize in your own software, which provides some common implementations of standard ideas.
- Many developers find that once they start using a framework like jQuery, there"s no going back to "pure" JavaScript because the framework offers so many useful shortcuts and brief ways of doing things.

In August 2005 jQuery founder John Resig was looking into how to better combine **CSS selectors with succinct JavaScript notation**.

- Within 1 year AJAXand animations wereadded
- Additional modules
  - jQuery UI extension
  - mobile device support
- Continues to improve.

jQuery Selectors

- When discussing basic JavaScript we introduced the getElementByID () and querySelector() selector functions in JavaScript.
- Although the advanced querySelector() methods allow selection of DOM elements based on CSS selectors, it is only implemented in newest browsers
- jQuery introduces its own way to select an element, which under the hood supports a myriad of older browsers for you.

- ✓ The relationship between DOM objects and selectors is so important in JavaScript programming that the pseudo-class bearing the name of the framework, **jQuery()**
- ✓ Is reserved for selecting elements from the DOM.
- ✓ Because it is used so frequently, it has a shortcut notation and can be written as **$()**

Basic Selectors

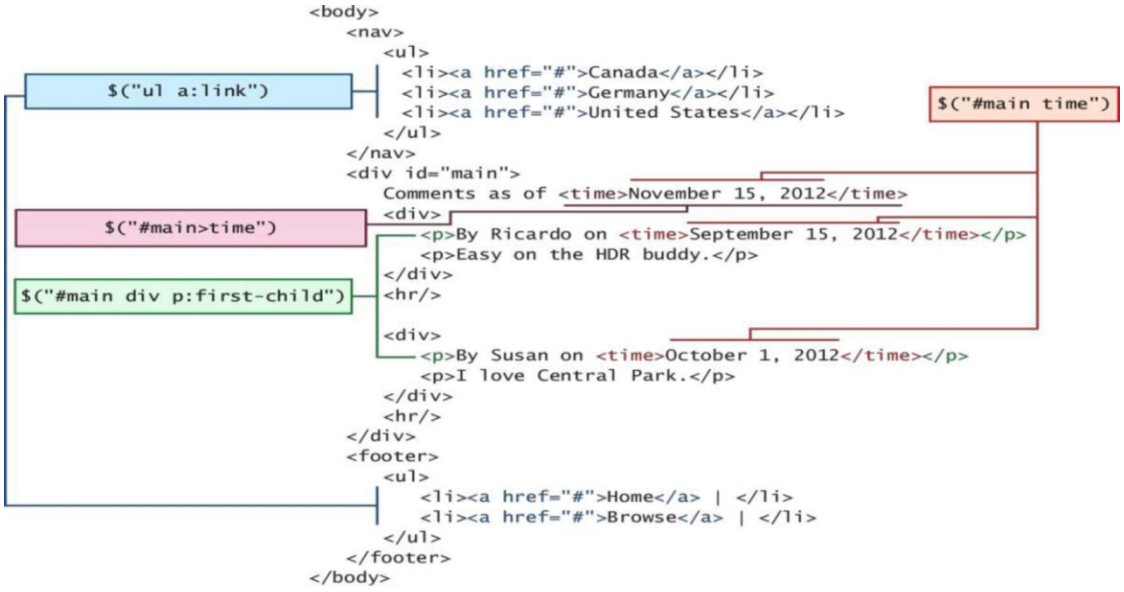
The four basic selectors are:

- $("*") Universal selector matches all elements (and is slow).
- $("tag") Element selector matches all elements with the given element name.
- $(".class") Class selector matches all elements with the given CSSclass.
- $("#id") Id selector matches all elements with a given HTML id attribute.
- For example, to select the single elementwith id="grab" you would write:
  var singleElement = $("#grab");
  Toget a set of all the <a> elements the selector would
  be: var allAs = $("a");
  These selectors replace the use of getElementById() entirely.

More CSS Selectors

In addition to these basic selectors, you can use the other CSSselectors:

- attribute selectors,
- pseudo-element selectors, and
- contextual selectors



Attribute Selector

Recall from CSS that you can select

- by attribute with square brackets
  - [attribute]
- Specify a value with an equals sign

- [attribute=value]
- Search for a particular value in the beginning, end, or anywhere Inside a string
  - [attribute^=value]
  - [attribute$=value]
  - [attribute*=value]

Pseudo-Element Selector

Pseudo-element selectors are also from CSS and allow you to append to any selector using the colon and one of

- :link
- :visited
- :focus
- :hover
- :active
- :checked
- :first-child, :first-line, and :first-letter

Selecting all links that have been visited, for example, would be specified with:

var visitedLinks = $("a:visited");

Contextual Selector

Contextual selectors are also from CSS. Recall that these include:

- descendant (space)
- child (>)
- adjacent sibling (+)
- and general sibling (~).

To select all <p> elements inside of elements you would write

var para = $("div p");

HTML Properties

- Many HTML tags include *properties* as well as attributes, the most common being the *checked* property of a radio button or checkbox.
- The prop () method is the preferred way to retrieve and set the value of a property although, attr() may return some (less useful) values.
- <input **class="meh"** type="checkbox" checked="checked">

- Is accessed by jQuery as follows:
- **var theBox = $(".meh"); theBox.prop("checked");** *// evaluates to TRUE* **theBox.attr("checked")**; *// evaluates to "checked"*

Changing CSS

- jQuery provides the extremely intuitive **css()**methods.
- Toget a css value use the css() method with 1 parameter:

$color = $("#colourBox").**css**("**background-color**"); *// **get** thecolor*

- Toset a CSSvariable use css() with two parameters: the first being the CSSattribute, and the second the value.

*// **set** color to red*

$("#colourBox").**css**("**background-color**", "**#FF0000**");

jQuery Listeners

In JavaScript, you learned why having your **listeners** set up inside of the window.onload() event was a good practice. With jQuery we do the same thing but use the $(document).ready () event

```
$(document).ready(function(){
 //set up listeners on the change event for the file items.
 $("input[type=file]").change(function(){
     console.log("The file to upload is "+ this.value);
  });
});
```

LISTING 15.6 jQuery code to listen for file inputs changing, all inside the document's ready event

Modifying the DOM

- The append () method takes as a parameter an HTML string, a DOM object, or a jQuery object. That object is then added as the last child to the element(s) being selected.

HTML Before

```
<div class="external-links">
     <div class="linkOut">
        funwebdev.com
     </div>
     <div class="linkIn">
        /localpage.html
     </div>
     <div class="linkOut">
        pearson.com
     </div>
<div>
```

jQuery append

```
$(".linkOut").append(jsLink);
```

HTML After

```
<div class="external-links">
     <div class="linkOut">
        funwebdev.com
<a href='http://funwebdev.com'
title='jQuery'>Visit Us</a>
     </div>
     <div class="linkIn">
        /localpage.html
     </div>
     <div class="linkOut">
        pearson.com
<a href='http://funwebdev.com'
title='jQuery'>Visit Us</a>
     </div>
<div>
```

- A more advanced technique might make use of the content of each div being modified. In that case we use a cal back function in place of a simple element.
- The wrap () method is a cal back function, which is called for each element in a set (often an array).
- Each element then becomes this for the duration of one of the wrap () function's executions, allowing the unique title attributes as shown in Listing15.12.

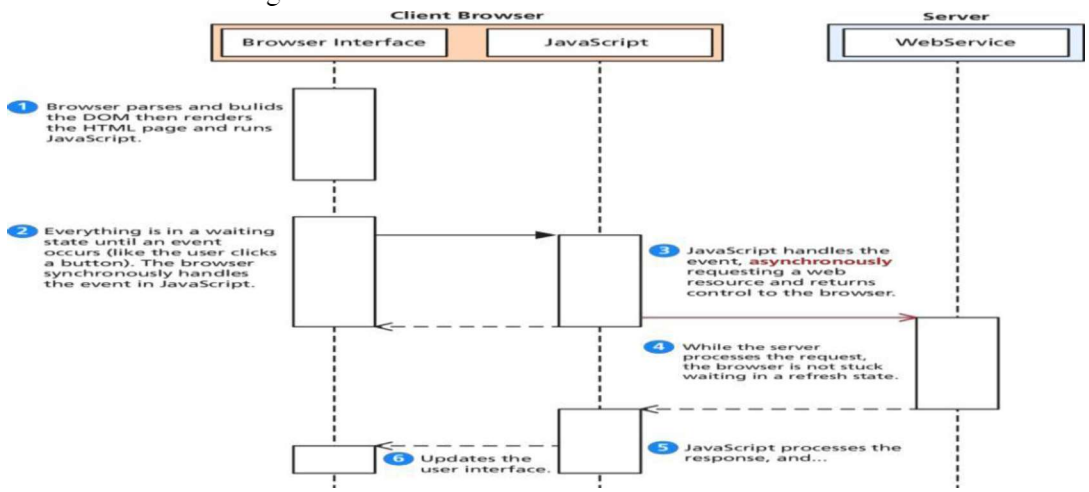
```
<div class="external-links">
 <div class="gallery">Uffuzi Museum</div>
 <div class="gallery">National Gallery</div>
 <div class="link-out">funwebdev.com</div>
</div>
```

```
<div class="external-links">
  <div class="galleryLink">
      <div class="gallery">Uffuzi Museum</div>
  </div>
   <div class="galleryLink">
      <div class="gallery">National Gallery</div>
  </div>
   <div class="link-out">funwebdev.com</div>
</div>
```

LISTING 15.10 HTML from Listing 15.9 modified by executing the wrap statement above

AJAX

- Asynchronous JavaScript with XML (AJAX) is a term used to describe a paradigm that allows a web browser to send messages back to the server without interrupting the flow of what's being shown in the browser.



- jQuery provides a family of methods to make asynchronous requests. We will start simple and work our way up.
- Consider the very simple server time example we just saw. If currentTime.php returns a single string and you want to load that value asynchronously

jQuery.get (**url [, data] [, success(data, textStatus, jqXHR)] [, dataType]**) **url** is a string that holds the location to send the request. **data** is an optional parameter that is a query string or a *PlainObject.* **success(data,textStatus,jqXHR)** is an optional *callback* function that executes when the response is received. **data** holding the body of the response as astring. **textStatus** holding the status of the request (i.e.,"success"). **jqXHR** holding a jqXHR object, described shortly. **dataType** is an optional parameter to hold the type of data expected from the server.

```
$.get("/vote.php?option=C", function(data,textStatus,jsXHR) {
  if (textStatus=="success") {
     console.log("success! response is:" + data);
  }
  else {
     console.log("There was an error code"+jsXHR.status);
  }
  console.log("all done");
});
```

LISTING 15.13 jQuery to asynchronously get a URL and outputs when the response arrives

All of the $.get() requests made by jQuery return a **jqXHR** object to encapsulate the response from theserver. In practice that means the data being referred to in the cal back from Listing 15.13 is actually an object with backward compatibility with XMLHttpRequest

- **Abort ()** stops execution and prevents any call-back or handlers from receiving the trigger to execute.
- **getResponseHeader()** takes a parameter and gets the current value of that header.
- **readyState** is an integer from 1 to 4 representing the state of the request. The values include 1: successful call open() 2: sending, 3: response being processed, and 4: completed.
- **responseXML**and/or **responseText** the main response to the request.
- **setRequestHeader(name, value)** when used before actually instantiating the request allows headers to be changed for the request.
- **status** is the HTTPrequest status codes (200 =ok)

- **statusText** is the associated description of the statuscode.

POST requests

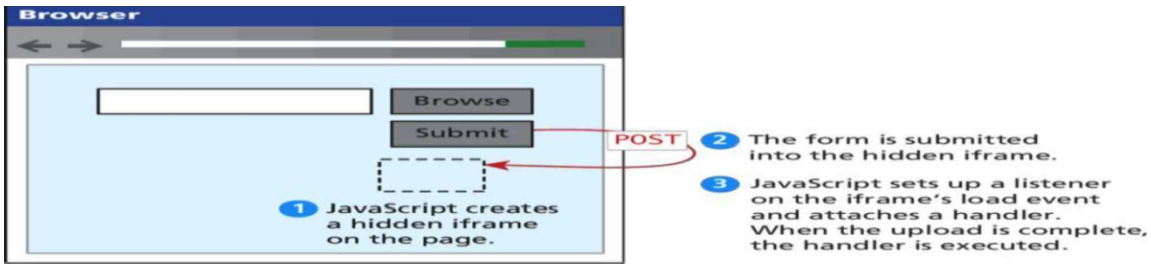
- POST requests are often preferred to GET requests because one can post an unlimited amount of data, and because they do not generate viewable URLs for each action.
- GET requests are typically not used when we have forms because of the messy URLs and that limitation on how much data we can transmit.
- With POST it is possible to transmit files, something which is not possible with GET.
- The HTTP 1.1 definition describes GET as a **safe method meaning** that they should not change anything, and should only read data.
- POSTs on the other hand are not safe, and should be used whenever we are changing the state of our system (like casting avote). get() method.
- POST syntax is almost identical to GET.
- jQuery.**post** (url [, data] [, success(data, textStatus, jqXHR)] [, dataType])
- If we were to convert our vote casting code it would simply change the first line from
- var jqxhr = $.get("/vote.php?option=C"); to
  var jqxhr = $.post("/vote.php", "option=C");

CORS

- Since modern browsers prevent cross-origin requests by default (which is good for security), sharing content legitimately between two domains becomes harder.
- Cross-origin resource sharing (CORS) uses new headers in the HTML5 standard implemented in most new browsers.
- If a site wants to allow any domain to access its content through JavaScript, it would add the following header to all of its responses.
- Access-Control-Allow-Origin:
- A better usage is to specify specific domains that are allowed, rather than cast the gates open to each and every domain. To allow our domain to make cross site requests we would add the header:
- Access-Control-Allow-Origin: www.funwebdev.com
- Rather than the wildcard*.

ASYNCHRONOUSFILE TRANSMISSION

The original workaround to allow the asynchronous posting of files was to use a hidden <iframe> element to receive the posted files.



```
<form name="fileUpload" id="fileUpload" enctype="multipart/form-data"
      method="post" action="upload.php">
<input name="images" id="images" type="file" multiple />
<input type="submit" name="submit" value="Upload files!"/>
</form>
```

LISTING 15.17 Simple file upload form

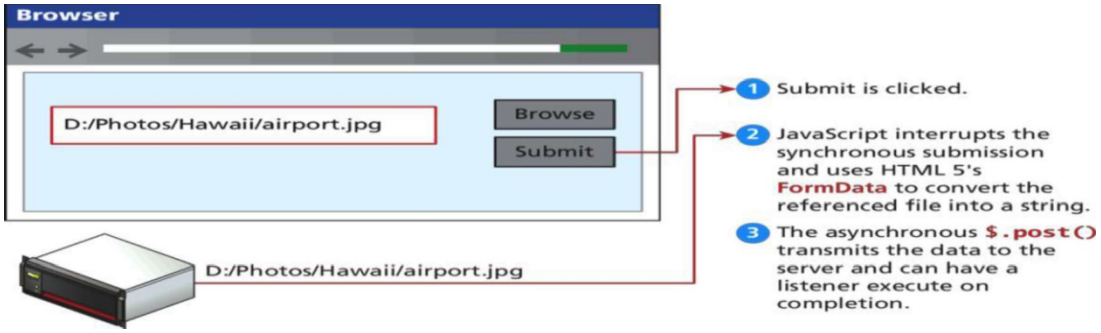
```
$(document).ready(function() {
  // set up listener when the file changes
  $(":file").on("change",uploadFile);
  // hide the submit buttons
  $("input[type=submit]").css("display","none");
});

// function called when the file being chosen changes
function uploadFile () {
  // create a hidden iframe
  var hidName = "hiddenIFrame";
  $("#fileUpload").append("<iframe id='"+hidName+"' name='"+hidName+"'
  style='display:none' src='#' ></iframe>");

  // set form's target to iframe
  $("#fileUpload").prop("target",hidName);
  // submit the form, now that an image is in it.
  $("#fileUpload").submit();
  // Now register the load event of the iframe to give feedback
  $('#'+hidName).load(function() {
  var link = $(this).contents().find('body')[0].innerHTML;
  // add an image dynamically to the page from the file just uploaded
  $("#fileUpload").append("<img src='"+link+"' />");
  });
}
```

LISTING 15.18 Hidden iFrame technique to upload files

Using the **FormData** interface and File API, which is part of HTML5, you no longer have to trick the browser into posting your file data asynchronously.



- When we consider uploading multiple files, you may want to upload a single file, rather than the entire form every time. To support that pattern, you can access a single file and post it by appending the raw file to a FormData object.
- The advantage of this technique is that you submit each file to the server asynchronously as the user changes it; and it allows multiple files to be transmitted at once.

```
var xhr = new XMLHttpRequest();
// reference to the 1st file input field
var theFile = $(":file")[0].files[0];
var formData = new FormData();
formData.append('images', theFile);
```

LISTING 15.20 Posting a single file from a form

```
var allFiles =  $(":file")[0].files;
for (var i=0;i<allFiles.length;i++) {
  formData.append('images[]', allFiles[i]);
}
```

LISTING 15.21 Looping through multiple files in a file input and appending the data for posting

ANIMATION

The hide () and show () methods can be called with no arguments to perform a default animation. Another version allows two parameters: the duration of the animation (in milliseconds) and a cal back method to execute on completion.

The fadeIn() and fadeOut() shortcut methods control the opacity of an element. The parameters passed are the duration and the cal back, just like hide () and show (). Unlike hide () and show (), there is no scaling of the element, just strictly control over the transparency.

As you may have seen, the shortcut methods come in pairs, which make them ideal for toggling between a shown and hidden state. Using a toggle method means you don't have to check the current state and then conditionally call one of the two methods;

- To toggle between the visible and hidden states you can use the **toggle ()** methods.
- To toggle between fading in and fading out, use the **fade Toggle()** method
- To toggle between the two sliding states can be achieved using the **slideToggle()**method.

Raw Animation

- The animate() method has several versions, but the one we will look at has the following form:
- .animate(properties, options);
- The properties parameter contains a Plain Object with all the CSSstyles of the final state of the animation.
- The options parameter contains another Plain Object with any of the following options set:
- Always is the function to be called when the animation completes or stops with a fail condition. This function will always be called (hence the name).
- Done is a function to be called when the animation completes.
- Duration is a number controlling the duration of the animation.
- Fail is the function called if the animation does not complete.
- Progress is a function to be called after each step of the animation.

- Queue is a Boolean value telling the animation whether to wait in the queue of animations or not. If false, the animation begins immediately.
- Step is a function you can define that will be called periodically while the animation is still going.
- Advanced options called easing and special Easing allow for advanced control over the speed of animation.

In web development, easing functions are used to simulate that natural type of movement. They are mathematical equations that describe how fast or slow the transitions occur at various points during the animation.

Included in jQuery are

- linear
- swing
- Easing functions.

BACKBONEMVC FRAMEWORKS

- Backbone is an MVC framework that further abstracts JavaScript with libraries intended to adhere more closely to the MVC model.

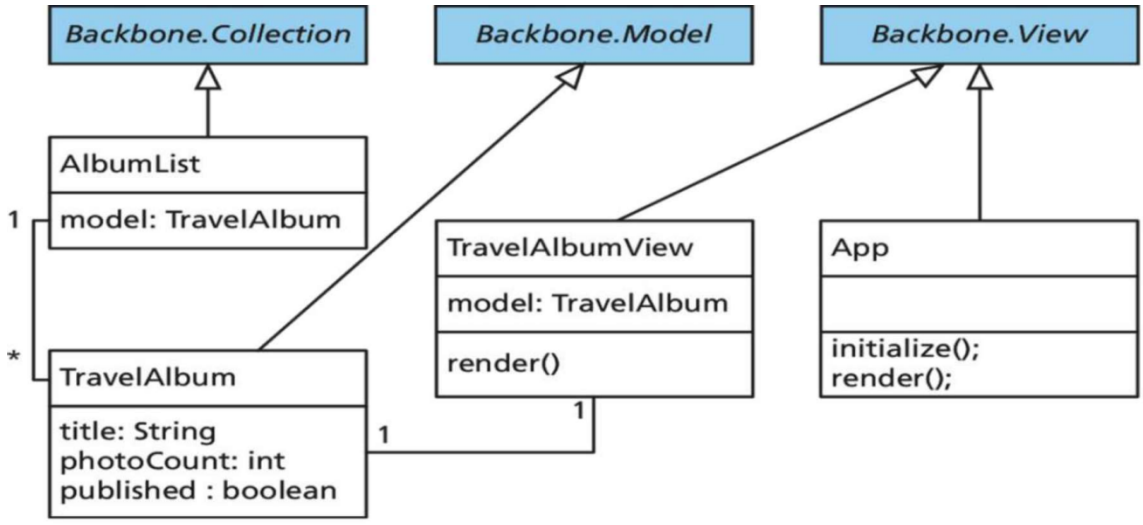
  Include with:

  <script src="underscore-min.js"></script>

  <script src="backbone-min.js"></script>

- In Backbone, you build your client scripts around the concept of **models**.
- Backbone.js defines **models** as *the heart of any JavaScript application, containing the interactive data as well as a large part of the logic surrounding it: conversions, validations, computed properties, and access control.*
- The Models you define using Backbone must *extend* Backbone Model

- ✓ In addition to models, Backbone introduces the concept of **Collections**, which are normally used to contain lists of Model objects.
- ✓ These collections have advanced features and like a database can have indexes to improve search performance.
- ✓ A collection is defined by extending from Backbone's Collection object.



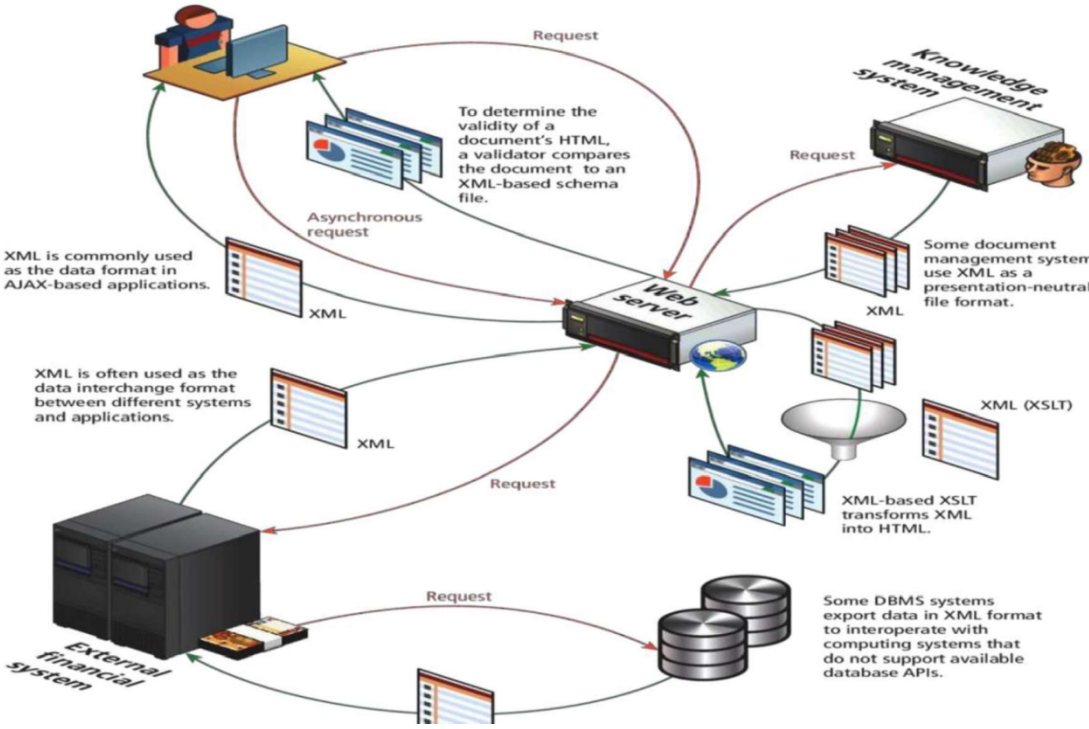

```
// Create a model for the albums
var TravelAlbum = Backbone.Model.extend({
   defaults:{
            title: 'NewAlbum',
            photoCount: 0,
            published: false
        },

        // Function to publish/unpublish
        toggle: function(){
            this.set('checked', !this.get('checked'));
        }
    });
```

LISTING 15.25 A PhotoAlbum Model extending from Backbone.Model

XML Processing and Web Services

- ❑ XML is a markup language, but unlike HTML, XML can be used to mark up any type of data.
- ❑ Benefits of XML data is that as plain text, it can be read and transferred between applications and different operating systems as well as being human-readable.
- ❑ XML is used on the web server to communicate asynchronously with the browser
- ❑ Used as a data interchange format for moving information between systems



Well Formed XML

For a document to be **well-formed XML**, it must follow the syntax rules for XML:

- Element names are composed of any of the valid characters (most punctuation symbols and spaces are not allowed) in XML.
- Element names can't start with number.
- There must be a single-root element. A **root element** is one that contains all the other elements; for instance, in an HTML document, the root element is<html>.
- All elements must have a closing element (or be self-closing).
- Elements must be properly nested.
- Elements can contain attributes.
- Attribute values must always be within quotes.
- Element and attribute names are case sensitive.
- A **valid XML** document is one that is well formed and whose element and content conform to a document type definition (DTD) or its schema.
- A DTD tells the XML parser which elements and attributes to expect in the document as well as the order and nesting of those elements.
- A DTD can be defined within an XML document or within an external file.

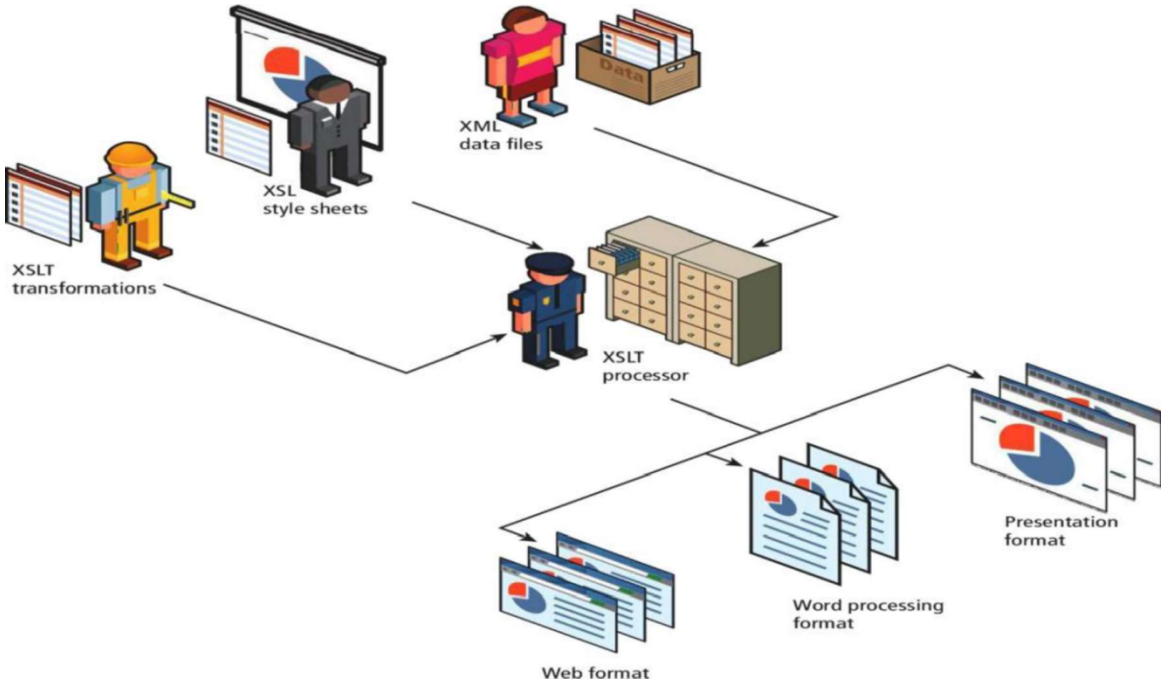
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE art [
<!ELEMENT art (painting*)>
<!ELEMENT painting (title,artist,year,medium)>
<!ATTLIST painting id CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT artist (name,nationality)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT nationality (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT medium (#PCDATA)>
]>
<art>
...
</art>
```

LISTING 17.2 Example DTD

- The main drawback with DTD is that they can only validate the existence and ordering of elements. They provide no way to validate the values of attributes or the textual content of elements.
- For this type of validation, one must instead use XML schemas, which have the added advantage of using XML syntax. Unfortunately, schemas have the corresponding disadvantage of being long-winded and harder for humans to read and comprehend; for this reason, they are typically created with tools.

XSLT

XSLT is an XML-based programming language that is used for transforming XML into other document formats



XPath

- **XPath** is a standardized syntax for searching an XML document and for navigating to elements within the XML document
- XPath is typically used as part of the programmatic manipulation of an XML document in PHP and other languages
- XPath uses a syntax that is similar to the one used in most operating systems to access directories.

XML PROCESSING

XML processing in PHP, JavaScript and other modern development environments is divided into two basic styles:

- The **in-memory approach**, which involves reading the entire XML file into memory into some type of data structure with functions for accessing and manipulating the data.
- The **event or pull approach**, which lets you pull in just a few elements or lines at a time, thereby avoiding the memory load of large XML files.
- All modern browsers have a built-in XML parser and their JavaScript implementations support an in-memory XML DOM API.

You can use the already familiar DOM functions such as

- getElementById(),
- getElementsByTagName()
- createElement()

To access and manipulate the data.

```
<script>
if (window.XMLHttpRequest)  {
  // code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
}
else  {
  // code for old versions of IE (optional you might just decide to
  // ignore these)
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
// load the external XML file
xmlhttp.open("GET","art.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;

// now extract a node list of all <painting> elements
paintings = xmlDoc.getElementsByTagName("painting");
if (paintings) {
    // loop through each painting element
    for (var i = 0; i < paintings.length; i++)
    {
        // display its id attribute
        alert("id="+paintings[i].getAttribute("id"));

        // find its <title> element
        title = paintings[i].getElementsByTagName("title");
        if (title) {
            // display the text content of the <title> element
            alert("title="+title[0].textContent);
        }
    }
}
</script>
```

LISTING 17.5 Loading and processing an XML document via JavaScript

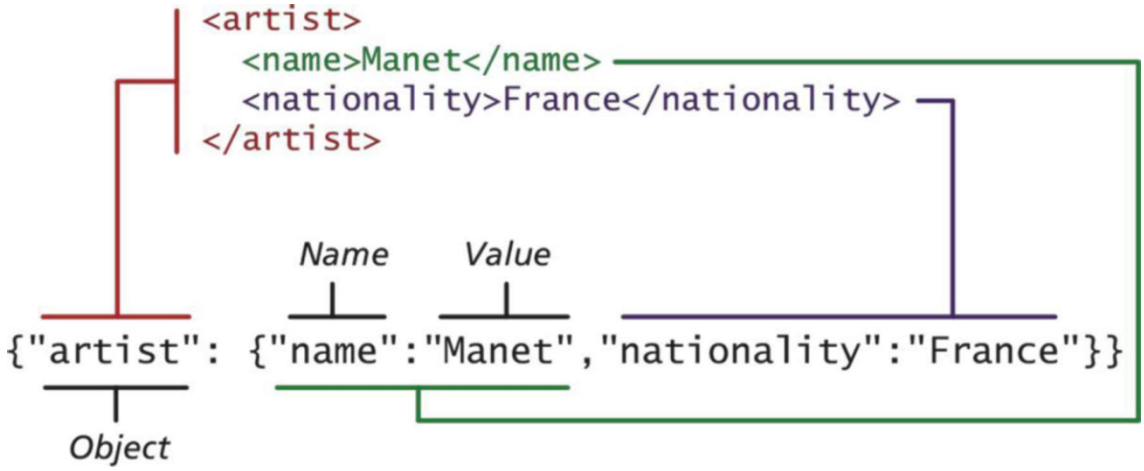
PHP provides several extensions or APIs for working with XML including:

- The **SimpleXML** extension which loads the data into an object that allows the developer to access the data via array properties and modifying the data via methods.
- The **XMLReader** is a read-only pull-type extension that uses a cursor-like approach similar to that used with database processing

JSON

- **JSON** stands for JavaScript Object Notation (though its use is not limited to JavaScript)
- Like XML, JSON is a data serialization format. It provides a more concise format than XML.

- Many REST web services encode their returned data in the JSON data format instead of XML.



OVERVIEW OF WEBSERVICES

- Web services are the most common example of a computing paradigm commonly referred to as service- oriented computing (SOC).
- A service is a piece of software with a platform- independent interface that can be dynamically located and invoked.
- Web services are a relatively standardized mechanism by which one software application can connect to and communicate with another software application using web protocols.
- They can provide interoperability between different software applications running on different platforms
- They can be used to implement service-oriented architecture (SOA)
- They can be offered by different systems within an organization as well as by different organizations

SOAP is the message protocol used to encode the service invocations and their return values via XML within the HTTP header.

- SOAP and WSDL are complex XML schemas
- akin to using a compiler: its output may be complicated to understand
- The enthusiasm for SOAP-based web services had

cooled. REST stands for Representational State Transfer.

- REST full web service does away with the service description layer, and needs no separate protocol for encoding message requests and responses.
- It simply uses HTTP URLs for requesting a resource/object (and for encoding input parameters).
- The serialized representation of this object, usually an XML or JSON stream, is then returned to the requestor as a normal HTTP response.

- REST appears to have almost completely displaced SOAP services.

Identifying and Authenticating Service Requests

Most web services are not open. Instead they typically employ one of the following techniques:

- **Identity**. Each web service request must identify who is making the request. **Authentication**. Each web service request must provide additional evidence that they are who they say they are
- Some web services are providing private/proprietary information or are involving financial transactions.
- In this case, these services not only may require an API key, but they also require some type of user name and password in order to perform an authorization.
- Many of the most well-known web services instead make use of the O Auth standard.